A SIMULATED MICRO-PROGRAMMED COMPUTER UTILIZING THE GRAPHIC DISPLAY OF AN IBM 360

by

Alan Wencil Frelich

United States
Naval Postgraduate School

THESIS

A SIMULATED MICRO-PROGRAMMED
COMPUTER UTILIZING THE GRAPHIC
DISPLAY OF AN IBM 360

by

Alan Wencil Frelich

and

Michael Charles Roth

December 1969

A Simulated Micro-Programmed Computer Utilizing the Graphic Display of an IBM 360

by

Alan Wencil Frelich
Lieutenant, United States Navy
B.S., United States Naval Academy, 1961

and

Michael Charles Roth
Lieutenant, United States Navy
B.S., United States Naval Academy, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1969

ABSTRACT

A small hypothetical computer was designed and simulated using a 2250 Display Unit operating on an IBM 360/67 computer. The hypothetical computer features a micro-programming capability which allows the user to design his own machine language structure for any particular application. The 2250 console is set up to simulate the operator's console of the hypothetical machine, with the CRT being used to display selected portions of memory and registers. For ease of use, a compiler is included in the system to allow the user to write his microprograms in a higher level language.

Discussions of the compiler, the interpreter, and the programming and creation of the proper type of graphic display are included. A set of operating instructions for the hypothetical computer enables the user to program and operate the machine without the need to be familiar with the 2250 display unit beforehand.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

As computers get larger and more sophisticated in their operations, the users (in our case, the NPGS students) are physically kept farther and farther away from the machine. It was not too long ago that in order to run a program the user had to know how to run the computer. Today an operating system runs the machine, and the user may not even see the computer he is working with; or at best, it is just a jumble of large boxes and flashing lights behind a plate glass window.

A. STATEMENT OF THE PROBLEM

To teach students what a computer is all about, smaller, more intimate computers are needed. Very often though, when in competition for a limited number of dollars, the small computer cannot be justified merely to enable some students to push the buttons themselves. A reasonable alternative in this situation is the simulation of the small computer using the large, sophisticated machine at hand, along with a graphics display terminal. With such a simulated system, the inner elements (memory locations and registers) of the hypothetical small computer can be displayed. The user can sit down in front of the display console, and using the alphameric keyboard and function keys provided, can operate the hypothetical computer as though it were a unique system only remotely connected to the "impersonal monster behind the plate glass window." This project concerns the design and usage of such a system utilizing an IBM 360/67 computer and an IBM 2250 Display Unit Model 1.

It would not be too difficult to design a specific computer and it's associated language and simulate that. This concept would be useful to a certain degree but would tend to stifle the full development of the user's own skills. By including a micro-programming capability however, the hypothetical computer becomes a framework, or set of fairly broad guidelines within which the user is free to design for himself. Each user can essentially define his own computer by defining his own repertoire of machine instructions. He can determine just which operations will be allowed and how these operations will be performed.

B. USE OF THE SYSTEM

The hypothetical computer described here may be used in a variety of ways. The user can define the structure of his computer through his micro-program, in either a higher level language which will be compiled into object code, or directly in object code. In either case he may then write machine language programs appropriate to his self-defined computer. The user may also be given an existing system by an instructor and only be required to write machine language programs which fit that computer. The Programmed Function Keys and the alphameric keyboard of the 2250 display unit have been set up to simulate the operator's console of the hypothetical computer. Through the proper use of the console and the control cards within his input to the hypothetical computer, the user has absolute control over all phases of the operation of his computer. The system is constructed so that a user can actually learn about computers and computing by gradually involving himself to an ever greater degree in the programming and operation of the hypothetical machine.

C. STRUCTURE OF THE SYSTEM

The hypothetical computer system is a FORTRAN program utilizing the graphic subroutine package (GSP) supplied by IBM. References 1, 2 and 3 were used extensively in writing the program. The program is modular in structure with the initialization, the compiler, the interpreter, and the Programmed Function Key control section comprising the modules. The modular construction facilitates the various degrees of involvement in the operation of the hypothetical computer.

D. ORGANIZATION OF THE REMAINDER OF THE PAPER

Section II describes the simulated microprogrammed computer in detail. Main memory, the general purpose registers, and the use of micro-memory in conjunction with the microprogram and the micro-interpreter are explained there.

Section III contains a description of the higher-level language and includes an example of a user-defined machine which is explained in detail. The object code generated by the compiler is also discussed and examples are given.

Section IV describes the 2250 display unit and includes a discussion of how the 2250 was programmed for the hypothetical computer. The operating instructions for the hypothetical computer are detailed in Section V. Concluding remarks appear in Section VI. Two additional examples are shown in the appendices, followed by sample program output and program listings.

II. THE MICROPROGRAMMED COMPUTER

Much interest has been given to programming computers using languages ranging from machine language to natural human languages. Not until recently, however, has there been much interest in designing systems with program control below the machine language level. Wilkes [4] noted that the machine instructions of a typical digital computer comprised a number of more elementary operations such as the transfer of information between registers or between registers and storage, shifting, adding, etc. He likened the execution of these elementary operations of a machine instruction to the execution of individual instructions in a machine program. Wilkes referred to these elementary operations as micro-instructions and is generally acknowledged as having coined the term "microprogramming".

In contrast to a typical digital computer in which all of the operations required for the execution of a machine instruction are hardware-logic wired, a microprogrammed machine uses preprogrammed micro-orders stored in memory to control the logic of the data paths. These micro-orders cause the machine to open gates, trigger flip-flops, set switches and to perform other elementary circuit functions.

Consider, as an example, the machine instruction

25100050

This instruction might say "take the contents of memory location 50, add it to the contents of register one, and place the sum back in register one". The execution of this machine instruction would require the execution of a number of more elementary operations such as

1. Gate "50" into the memory address register

2. Call the contents of location 50 as specified in the address register and gate the contents to the adder

3. Gate "1" to the general purpose register address register

4. Call the contents of register one as specified in the address register and gate the contents to the adder

5. Move the contents of the adder back to register one.

Suppose each of these elementary operations could in turn be controlled by execution of an instruction. Then a collection of these lower-level instructions (micro-orders) would have the same effect as one machine instruction. Thus a machine instruction could be designed by the appropriate selection of a subset from the set of available micro-orders.

Since the microprogrammed (stored logic) concept allows the instantaneous logical organization of the computer to be altered, the need for an elaborate hardware package of fixed construction for all uses and users is precluded. The stored logic concept is of special interest to areas where size and weight of the hardware would be critical and/or where the given application is specific. Ideally, a microprogram facility would give the casual user direct access to the micro-orders and to their order of performance. However, adequately mastering the techniques of microprogramming requires a thorough understanding of the micro-orders available, the internal logic of the computer, and the circuitry involved, which almost eliminates the ordinary user. Attempts have been made through the use of plugboard or switching control tape to make the microprogram physically easy to set up. However, this approach has been objectionable on aesthetic grounds and it would appear to be a backward step in computer development. There has also been talk of providing higher-level languages for microprogramming. Although compiling introduces inefficiencies in the microprogram, this might be the route for the

ordinary user. References 5-10 describe a number of microprogrammed computers. The simulated microprogrammed computer used here will be described in detail in this section of the report.

A. GENERAL DESCRIPTION OF THE COMPUTER

The microprogram for the simulated microprogrammed computer designed here can be written in either a higher-level language or in a microprogram object code. The microprogram logic for execution of the machine instructions is stored in micro-memory at compile or read-in time. Any type of machine instruction can be made up out of the available micro-orders and stored sequentially in memory as a micro-routine. The microprogram consists of a series of micro-routines, one for each different machine instruction defined by the user, and a master micro-routine to effect branching to the appropriate routine for the current instruction. Each instruction in the machine program can be thought of as a macro call with an operation code making the call and the remainder of the instruction supplying the parameters.

The basic system consists of main memory storage, micro-memory storage and a bank of general purpose registers. The micro-memory with its associated sequencing capability is the heart of the micro-programmed system. It contains a number of special use locations which serve as pseudo registers. The user's machine programs and associated data are stored in main memory while the microprogram and its associated control data are stored in micro-memory. Storage and retrieval accesses with main memory and the general purpose registers is effected through the use of address and buffer registers.

The basic control word used within the system is the micro-order. The micro-order consists of eight decimal digits which at interpretation

time is divided into two four-digit addresses called the A address and the B address, as shown in Figure 1. The execution of one micro-order

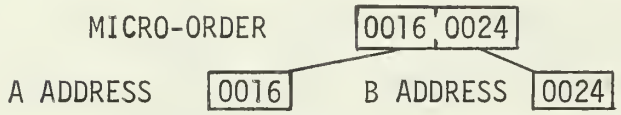

FIGURE 1. Micro-order format.

consists of two phases. The first phase is a fetch operation in which the A address is used to determine the location of a datum for transfer to a special register called the transfer register (T REG). The second phase is a store operation in which the B address is used to determine where the contents of the T REG are to be placed. The execution of one micro-order constitutes a single cycle through the micro-interpreter, and one or more cycles complete a single machine language instruction. During the course of one cycle, certain side effects may occur. These side effects are explained in detail later. In most cases, however, the only result of the execution of one micro-order is the movement of a datum from one micro-memory location to another.

B. MICRO-MEMORY

Micro-memory contains 999 words of storage addressed numerically 1 through 999. The micro-memory locations are referred to by L1, L2 ··· L999. At compile time or at load time, depending on the method of micro-program input, the users microprogram is stored in micro-memory in ascending sequential cells starting at location 201. A breakdown of the different areas of use in memory is depicted in Figure 2.

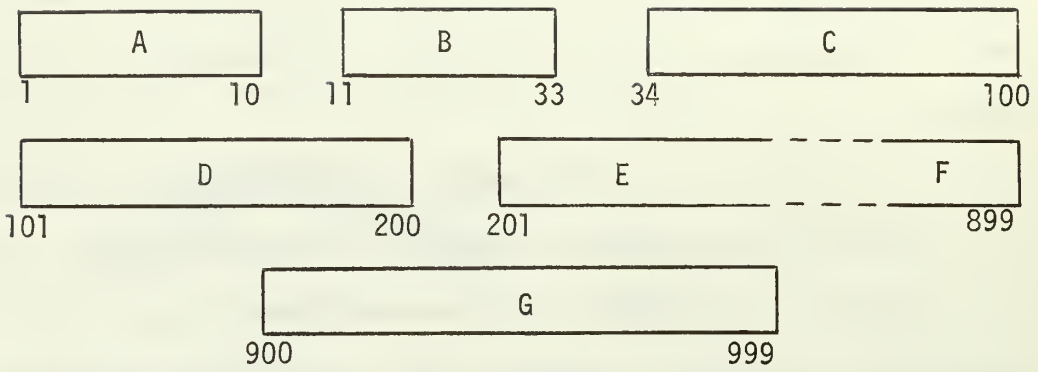

FIGURE 2. Micro-memory areas of allocation

Area A contains ten words of memory whose use can be explicitly and arbitrarily assigned by the user. Area B contains special storage whose use is controlled by the micro-interpreter to perform specific operations. Area C is a working storage space whose locations may be addressed directly by the user and which serve no special functions except as programmed by the user. Area D functions as a buffer area. Locations 101 through 180 inclusive are used as an input/output buffer during the read and write operations. Locations 181 through 200 are convenient to use for intermediate data manipulation using GET and PUT operations. Area E contains the user's microprogram and fills upward numerically from location 201. Area F contains the constant table for the higher-level language compilation and fills downward numerically from location 899. The upper limit of area E is one less than the lower limit of area F, and vice versa. Area G is where the compiler puts branching instructions to be used by the master routine to effect linking to the appropriate micro-routine for each machine language instruction.

1. The User-Controlled Area (Locations L1-L10)

These locations may be assigned in the user's higher-level language definition of the microprogram. They may be assigned arbitrarily to contain any information considered necessary or pertinent to implementation of the user's machine. Items of interest might be a main memory location counter pointing to the next machine instruction to be executed, an instruction register containing the current instruction, or might be certain parts of the machine instruction such as an operation code, a register number, an index register, a tag, addresses (direct or indirect), or a condition code. For example, consider one design for a single-address machine instruction as illustrated in Figure 3(a).

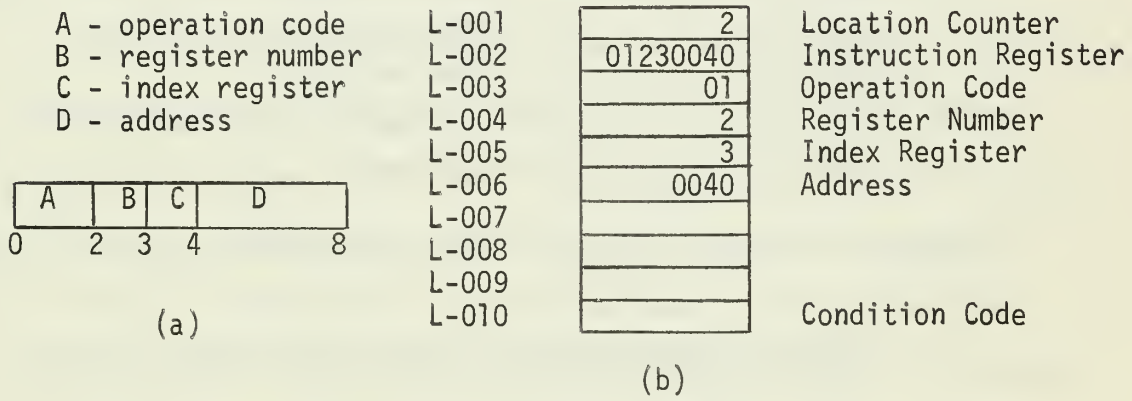

FIGURE 3. Design for a single-address machine instruction.

The instruction must be broken up into the four parts in order that operations can be performed individually on each part. The user may desire to store the various items in the locations as shown in Figure 3(b). A method for separating the different fields of the instruction and assigning them to the desired locations will be explained later.

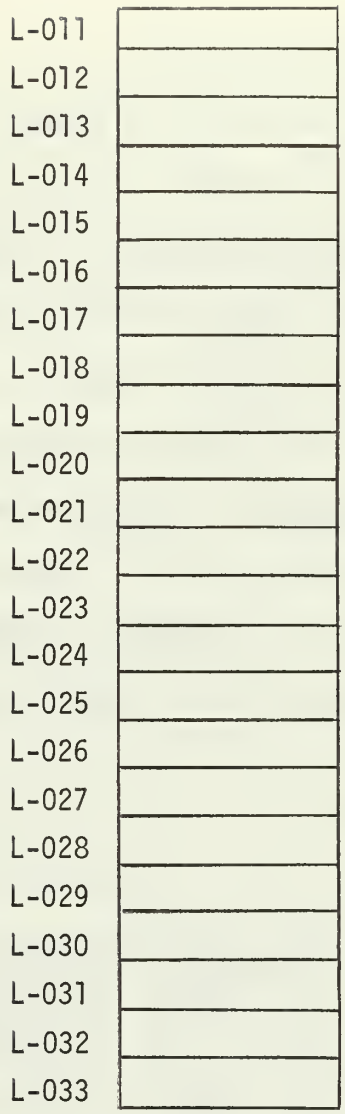

FIGURE 4. Pictorial layout of the interpreter-controlled locations

2. The Interpreter-Controlled Area (Locations L11-L33)

The interpreter-controlled locations are depicted in Figure 4. Locations L11-L15 are used by the "hardware" for basic control of the interpretation cycle.

L11. The micro-memory location counter is incremented at the beginning of each cycle and contains the location of the micro-order to be executed next in sequence. Its contents are modifiable under program control for branching and linking.

L12-L14. The micro-order register (L12) holds the current micro-order being executed and is loaded at the beginning of each cycle with the contents of the micro-memory cell whose address is in L11. At interpretation time, the micro-order stored in L12 is separated into two parts, the A address and the B address. The A address is stored in the fetch control register (L13), which is used to control retrieval of data from micro-memory. The B address is placed in the store control register (L14), which is used to control storage of data in micro-memory.

L15. A micro-memory condition code is set and stored in L15 whenever locations L24-L31 (the arithmetic registers) are used as the B address of a micro-order. A condition code of "1" is set if the contents of the referenced location is negative. A "2" is set if the contents equal zero, and a "3" is set if the contents are positive.

Locations L16-L33 have special functions associated with them. Some produce certain "side effects" when they are referenced, while others contain the necessary information to control these side effects.

L16-L17. Whenever L16 appears as the A address of a micro-order, a datum is fetched from a general purpose register using the address found in L17. This datum is then stored in L16 and also placed in the T REG. An example showing the contents of a number of machine locations after the execution of a typical micro-order in L357 appears in Figure 5. Whenever L16 appears as the B address of a micro-order, the datum in the T REG is moved to L16 and is also stored in the general purpose register whose address is found in L17.

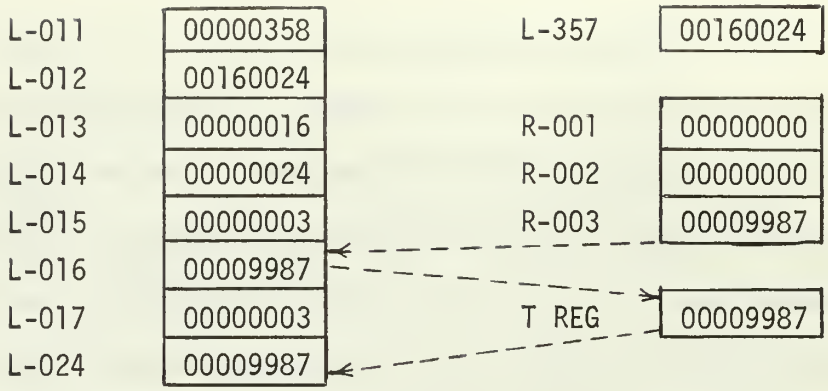

FIGURE 5. An example illustrating L16 used as the A address of a micro-order.

L18-L19. Whenever L18 appears as the A address of a micro-order, a datum is fetched from main memory using the address found in L19. The datum is stored in L18 and also placed in the T REG. Whenever L18 appears as the B address of a micro-order, the datum in the T REG is moved to L18 and is also stored in main memory at the location specified in L19.

L20. The read-write control register is used to determine whether a read or a write operation is to be executed. If L20 is used as the A address, a card is read and its contents are loaded into the I/O buffer in EBCDIC format with card columns 1-80 corresponding to buffer locations 101-180. For example, suppose the card being read contained the number 436 in card columns 3-5. Then the contents of the I/O buffer would look like:

Location	101	102	103	104	105	106	....	180
Contents	64	64	244	243	246	64		64

Each number on the card is converted and inserted in the buffer in EBCDIC representation (e.g., a decimal number in the range 0 to 255). If L20 is used as the B address, the contents of buffer locations L101-L180 are printed as a single line of output.

L21-L23. Whenever L21 is used as the A address, each of the data stored in ascending micro-memory locations starting with the location specified in L22 up to and including the location specified in L23, are converted from EBCDIC format to decimal digits, combined into an integer number and stored in L21. Non-numeric characters other than a minus sign are ignored during the conversion. The result in L21 is then placed in the T REG. If L21 is used as the B address, the procedure is reversed. That is, the number in the T REG is first moved to L21. It is then separated into individual decimal digits which are converted to EBCDIC format. The EBCDIC representations of the decimal digits are finally stored in the buffer locations as specified by the addresses in L22 and L23. The rightmost digit is right adjusted in the highest location, and leading blanks and a minus sign are inserted if necessary.

L24-L31. Whenever L24 appears as the B address, the datum in the T REG is placed in L24 and L25. This procedure serves to initialize the add register (L25). Whenever L25 appears as the B address, the datum in the T REG is added to the current contents of L25, and the result is stored in L25. In this case, the contents of L24 remain unchanged. The L26-L27, L28-L29, and L30-L31 pairs function the same as the L24-L25 pair to do subtraction, multiplication, and division, respectively. When each location, L24-L31, appears as the B address, the appropriate condition code is set and stored in L15.

L32-L33. Whenever L32 appears as the A address of a micro-order, a datum is fetched from micro-memory using the address found in L33. The datum is stored in L32 and also placed in the T REG. When L32 appears as the B address of a micro-order, the datum in the T REG is moved to

L32 and is also stored in micro-memory at the location specified in L33. L32 and L33 are convenient to use when a micro-memory location is specified "indirectly" (e.g. L(L35)).

3. The Microprogram (Locations L201-L999)

The microprogram consists of a master routine and a set of micro-routines used to perform the tasks required by the instruction repertoire of the selected machine language. The master routine may contain any number of special function sequences, two of which must be a get-next-instruction (GNI) sequence and a linking sequence. The GNI sequence is required in order to access each individual instruction from main memory and to separate different parts from the instruction. The GNI sequence also controls a location counter which is used to access successive instructions from main memory. The linking sequence determines which instruction of the repertoire was accessed in order to provide branching in micro-memory to the micro-routine associated with the accessed instruction. The linking sequence accomplishes branching by inserting a new micro-memory address into the micro-memory location counter. This is done via a transfer link in the branching table which accesses an address in the constant table.

After branching to the selected micro-routine has been effected, the sequence of micro-order execution within the routine is controlled by the micro-memory location counter. The address in the location counter, however, may be changed by the logic of the micro-routine to control branching within the routine. Each routine may use parts of the instruction as operands to access main memory or the general purpose registers. Data transfer within micro-memory is controlled directly by the addresses of the micro-order. Each micro-routine associated with instruction execution returns control to the GNI sequence of the master routine when its task is completed.

The simulator with user microprogramming lends flexibility to allow the design of a large and varied instruction repertoire. The microprogram may be written in a higher-level language or directly in object coded format. Both methods offer a challenge in ingenuity and creativity in designing an instruction set, however microprogramming in object code further challenges the user in writing efficient and optimal coding.

C. MAIN MEMORY

Main memory contains 1000 words of storage addressed numerically 0 through 999. Main memory locations may be referred to by the prefix letter M followed by the address (e.g., M2 or M98). Each word can hold an 8-digit positive number or a 7-digit negative number. Associated with the main memory cells are the memory address register (L19) and the memory buffer register (L18). L19 holds the address of the datum of interest while L18 holds the datum to be recalled or receives the datum to be memorized. The two functions of memory are recall and memorization. A recall is initiated when L18 is used as the A address of a micro-order. A datum is then returned from the main memory location specified in L19. Memorization is initiated when L18 is used as the B address, whence a datum is then stored in main memory at the location specified in L19.

The users machine language program is stored in main memory. The basic machine language instruction consists of a series of decimal digits up to a maximum length of eight digits. The format of the instruction may be designed in any manner to suit the needs of the user. The instruction is decoded or broken up into separate parts

under microprogram control. The user must define the microprogram in order to separate each piece of information form the instruction and to store it in the proper micro-memory location where it can be operated upon by the interpreter.

D. THE GENERAL PURPOSE REGISTERS

The system contains 100 general purpose registers numbered R0 through R99. The registers may be used for indexing to implement address modification. Associated with the GPR's are a general purpose register buffer register (L16) and a general purpose register address register (L17), whose functions are analogous to L18 and L19.

III. THE HIGHER-LEVEL MICROPROGRAMMING LANGUAGE

The task of writing the microprogram directly in object-coded format is somewhat tedious. Therefore, the user may prefer to write the microprogram in a higher-level language which could be used by the compiler to generate the necessary object code. This section of the report defines the higher-level language used and gives an example of a higher-level language microprogram described in detail. Also included are some examples of the compiler-generated object code.

A. DEFINITION OF THE HIGHER-LEVEL LANGUAGE

Following is a BNF grammar of the language.

<microprogram> :: = <statement list> <group list> <end statement>

<group list> :: = <group> |<group list> <group>

<group> :: = <group head> |<group head> <statement list>

<group head> :: = <op code label> <statement>

<statement list> :: = <statement> |<statement list> <statement>

<statement> :: = <let statement >|<if statement>| <read statement>|
<write statement> |<get statement> |<put statement>|
<go to statement> |<return statement> |<stop statement>

<let statement> :: = LET <variable> = <expression>

<expression> :: = <operand> |<operand> <operator> <operand>

<operand> :: = <variable> |<constant>

<operator> :: = +|-|*|/

<if statement> :: = IF (<variable>) <number>, <number>, <number>

<read statement> :: = READ|READ (<parameter list>)

<write statement> :: = WRITE|WRITE (<parameter list>)

<get statement> :: = GET (<parameter list>)

<put statement> :: = PUT (<parameter list>)

<parameter list> :: = <operand>, <operand>, <variable>

<go to statement> :: = GO TO <number>

<return statement> :: = RETURN

<stop statement> :: = STOP

<end statement> :: = END

<op code label> :: = <number> { ≥ 1, ≤ 99 }

<variable> :: = <subscripted variable>| <nonsubscripted variable>

<subscripted variable> :: = <letter> (<subscript>)

<subscript> :: = L<number>| <constant>

<nonsubscripted variable> :: = <letter> <number>

<number> :: = <decimal digit> |<number> <decimal digit>

<decimal digit> :: = 0|1|2|···|9

<letter> :: = L|R|M

<constant> :: = <number> { ≥0, ≤99999999 }

The reserved words READ, WRITE, GET, PUT, LET, IF, GO TO, RETURN, STOP, and END are used by the compiler to determine the type of statement being compiled. (In each statement the reserved word must be followed by at least one blank.) The group list used in the BNF refers to the collection of micro-routines, each routine being called a group. The first statement of each routine must be preceeded by a label which is the numerical operation code. The last statement of a routine would usually be a return statement to return control back to the master routine. If the last statement is not a return statement then it must be a go to statement and the return statement must be elsewhere within

the routine. The group list must be followed by the END statement. The END statement tells the compiler that this is the last statement in the microprogram and that compilation can be terminated.

The parameter list in the read and write statements is optional. READ or WRITE without the parameter list simply results in the reading of a card into the buffer or the writing of the contents of the buffer into one line of output. If a parameter list is used, data is transferred from the buffer to a storage location after a read, or data is transferred from a storage location to the buffer before a write. The first two parameters in the list may be variables or constants, while the third parameter must be a variable.

B. A HIGHER-LEVEL LANGUAGE EXAMPLE

An example of a higher-level language microprogram is presented here.

Consider the machine instruction design as shown in Figure 1(a) on page 13 and segregation of information as shown in Figure 1(b). The operation codes and their corresponding operations are as follows:

1	Read
2	Write
3	Load
4	Store
5	Add
8	Divide
10	Compare
11	Branch on less than
14	Stop

A microprogram to describe the special set of instructions listed above is shown in Figure 6.

```
        LET L2 = M(L1)                    (1)
        LET L1 = L1 + 1                   (2)
        PUT (181,188,L2)                  (3)
        GET (181,182,L3)                  (4)
        GET (183,183,L4)                  (5)
        GET (184,184,L5)                  (6)
        GET (185,188,L6)                  (7)
        LET L6 = L6 + R(L5)               (8)
        LET L11 = L3 + 900               (10)

1       READ                             (11)
        LET L40 = 101                    (12)
        LET L41 = 101 + 4                (13)
        GET (L40,L41, M(L6)              (14)
        LET L40=L40 + 5                  (15)
        LET L6 = L6 + 1                  (16)
        LET R(L4) = R(L4) - 1            (17)
        IF (R(L4))9,9,3                  (18)
        RETURN                           (19)

2       WRITE (125,130,M(L6)             (20)
        RETURN                           (21)

3       LET R(L4) = M(L6)                (22)
        RETURN                           (23)

4       LET M(L6) = R(L4)                (24)
        RETURN                           (25)

5       LET R(L4) = R(L4) + M(L6)        (26)
        RETURN                           (27)
```

FIGURE 6. A Higher-Level Language Microprogram

```
 8    LET R(L4) = R(L4)/M(L6)          (28)
      RETURN                           (29)

10    LET L39 = R(L4) - M(L6)          (30)
      IF (L39) 3,5,7                   (31)
      LET L10 = 11                     (32)
      GO TO 8                          (33)
      LET L10 = 12                     (34)
      GO TO 8                          (35)
      LET L10 = 13                     (36)
      RETURN                           (37)

11    LET L38 = L10 - 11               (38)
      IF (L38) 4,3,4                   (39)
      LET L1 = L6                      (40)
      RETURN                           (41)

14    STOP                             (42)
      END                              (43)
```

FIGURE 6. (Continued)

The master routine consists of statements (1) through (10). The GNI sequence consists of statements (1) through (7). Statement (1) loads the instruction from the main memory location addressed by L1 into the instruction register (L2). Statement (2) increments the location counter (L1) by one. Statement (3) divides the contents of the instruction register into individual digits and inserts one digit in each of the buffer locations L181-L188. Statement (4) gets the digits stored in locations 181 and 182, combines them into an integer and stores the result in L3. Statements (5) through (7) get the remaining digits, combines them as indicated and store the results in L4, L5 and L6 respectively. The instruction is now broken up so that

its parts can be operated on or manipulated individually. Statement (8) computes the effective address by adding the contents of the index register to L6. In this case indexing is done in the master routine before branching to the micro-routine of the selected instruction. This has the effect of providing indexing for all of the instructions in the selected instruction set. If indexing were desired only in certain of the instructions, then statement (8) would not appear in the master routine but would appear in each of the micro-routines where indexing is desired. Statement (10) comprises the linking sequence. The base address of the branching table (900) is added to the operation code (L3). Suppose the current instruction was a load instruction. Then the next micro-order to be executed would be in location 903. The order stored in 903 by the compiler would load the base address of the "load" micro-routine into the location counter (L11) which in turn would cause branching to that routine.

The micro-routine consisting of statements (11) - (19) defines the read instruction. Statement (11) reads a card into the buffer. Statements (12) and (13) set the values of the parameters used in the following get statement. Statement (14) combines the data in the buffer locations specified by L40 and L41 and stores the result in the main memory location addressed by L6. For this read instruction, the contents of the register determines the number of data to be read from the same card and stored in successive memory locations. The parameters L40 and L41 determine where on the card the data are to be found. Consider the read instruction

01200040

Suppose R2 equals four. Then the first datum will be read from card columns 1-5 and stored in memory location 40. The second datum will

be read from card columns 6-10 and stored in memory location 41, and so on until four data are read and stored, at which time R2 will have been decremented to zero by statement (17). Statement (18) then causes branching to the ninth statement within the micro-routine. (Note that the numbers in an IF statement and in a GO TO statement refer to the position of another statement within the same micro-routine.) The return statement transfers control back to the master routine by loading into L11 the address of the first micro-order of statement (1).

The write micro-routine (statements (20) and (21)) inserts one datum from main memory into the buffer and prints the entire contents of the buffer. Any other data held in the buffer in locations other than 125-130 would also be printed. For example, had four numbers been read in previous to the write instruction, then these four numbers in locations 101-120 would also have been printed. To prevent the printing of any old or undesirable data held in the buffer from an earlier operation, it would be necessary to first clear the buffer before inserting any new data to be printed. Clearing of the buffer could be accomplished by a routine like the following:

```
LET L44 = 101                    (1)
LET L(L44) = 64                  (2)
LET L44 = L44 + 1                (3)
LET L45 = L44 - 180              (4)
IF (L45) 2,2,6                   (5)
WRITE (125,130,M(L6))            (6)
RETURN                           (7)
```

Statements (1) - (5) form a loop to set the contents of buffer locations 101-180 to blanks. L44 is used to indirectly specify the micro-memory location which is to be blanked. It is incremented by one each time through the loop until it equals 181, at which time a branch is taken to statement (6). Thus each buffer location is set equal to 64 which is the decimal equivalent of the EBCDIC representation of a blank.

The routines in statements (22) - (29) are self explanatory. The routine in statements (30) - (37) compares the contents of a register with the contents of a main memory location and sets a condition code of 11, 12, or 13 depending upon whether the result is negative, equal to zero, or positive. Statement (31) causes branching to statement (32), statement (34) or statement (36) where the applicable condition code is stored in L10. The branch-on-less-than routine checks to see if the condition code equals eleven. If L38 equals zero, then statement (40) is executed, otherwise the if statement causes branching to the return statement. If statement (40) is executed, then the main memory location counter is set equal to the address in L6.

The compiler for the higher-level language is a single pass compiler which generates the necessary object code as each statement of the higher-level-language is processed. The end statement signals the compiler to return control back to the function keyboard. A listing of the microprogram in object code for this example as generated by the compiler is contained on pages 83 and 84.

Figure 7(a) shows a machine program which is roughly equivalent to the simple FORTRAN-like program shown in Figure 7(b). This machine program could be executed by the hypothetical machine defined in Figure 6 on pages 26 and 27.

```
M-000    03200013
M-001    01200020
M-002    03300011
M-003    05130020
M-004    05300012
M-005    10300013
M-006    11000003
M-007    08100013
M-008    04100024
M-009    02000024
M-010    14000000
M-011    00000000
M-012    00000001
M-013    00000004
```

(a)

```
        READ (10) A, B, C, D
10      FORMAT (4I5)
        SUM = A + B + C + D
        AVE = SUM/4
        WRITE (15) AVE
15      FORMAT (I10)
        END
```

(b)

FIGURE 7. A machine language program and its corresponding FORTRAN-like program.

The machine as defined in Figure 6 is minimal in order to suit the machine language program illustrated here. However, the machine could be enlarged considerably by defining additional instructions.

Two additional examples of higher-level programming appear in the appendices. In Appendix A a machine using the "stack" concept is defined. The compiler output for this example appears on pages 85 and 86 . In Appendix B a simple BRAIN-2 machine is defined.

C. THE OBJECT-CODED MICROPROGRAM PRODUCED BY THE COMPILER

The following examples show the object code which would be generated by the compiler for typical higher-level language statements.

EXAMPLE 1

```
READ (L40, L41, M(L6))
      0020 0000      (1)
      0040 0022      (2)
      0041 0023      (3)
      0006 0019      (4)
      0021 0018      (5)
```

Micro-order (1) causes a card to be read. If the parameter list is not used, then micro-orders (2) - (5) are not generated. Micro-orders (2) and (3) transfer the contents of L40 and L41 into the buffer lower and upper address registers. Micro-order (4) loads L19 with the contents of L6 and micro-order (5) converts, combines and transfers data from the buffer into L18.

EXAMPLE 2

```
LET R(L4) = R(L4) + M(L6)
    0004 0017       (1)
    0016 0024       (2)
    0006 0019       (3)
    0018 0025       (4)
    0004 0017       (5)
    0025 0016       (6)
```

Micro-order (1) loads the contents of L4 into L17 and micro-order (2) transfers the contents of L16 into the add-initialize register. Micro-order (3) transfers the contents of L6 into L19 and micro-order (4) transfers the contents of L18 into the add register. Both micro-orders (2) and (4) set the condition code which is stored in L15. Micro-order (5) loads L17 and micro-order (6) transfers the result of the add operation to L16.

EXAMPLE 3

```
LET L(L35) = R4 - 10
    0348 0017       (1)
    0016 0026       (2)    L-348  0000 0004
    0349 0027       (3)    L-349  0000 0010
    0035 0033       (4)
    0027 0032       (5)
```

Micro-order (1) loads the constant "4" from the constant table into L17 and micro-order (2) transfers the contents of L16 to the subtract-initialize register. The constant "10" is then sent to the subtract register. In this example, the micro-memory location is addressed indirectly, therefore the address stored in L35 must be loaded into the micro-memory address register and the result in the subtract register transferred to the micro-memory buffer register. If L(L35) had been simply L35, then micro-orders (4) and (5) would simply be the one micro-order "0025 0035".

EXAMPLE 4

```
          IF (L38) 3, 5, 7
          L-236  0038 0024
          L-237  0015 0024
          L-238  0011 0025     L-350  0000 0223
          L-239  0025 0011     L-351  0000 0230
          L-240  0350 0011     L-352  0000 0243
          L-241  0351 0011
          L-242  0352 0011
```

Suppose the code is stored in the micro-memory locations as shown above with L350-L352 being part of the constant table. The micro-order stored in L236 transfers the contents of L38 into the add initialize register. This is done not to initiate an add operation but simply to set the condition code according to the contents of L38. L237 and L238 add the contents of the micro-memory location counter to the condition code and store the result in L11. The address in L11 then determines whether L240, L241, or L242 is executed next. A condition code of "1" would cause L240 to be executed next which in turn loads the contents of L350 into the location counter. L350 contains the address of the first micro-order of the third statement of the micro-routine which

contains the if statement. Condition codes of "2" and "3" would cause branching to locations 230 and 243 respectively. The overall effect is to cause branching within a micro-routine depending upon the value of the argument.

Returning control from a micro-routine back to the main routine may be executed by one micro-order which loads the address "201" into the micro-memory location counter. The stop instruction is recognized by the interpreter as a string of zeros (e.g., both the A address and the B address specify a no operation).

For the user to write the microprogram directly in object-coded format, he needs both the knowledge required to write the microprogram in the higher-level language and a thorough understanding of the interpreter-controlled locations (L11-L33). Besides writing the routines to define his machine, the user must also construct the constant table and the branching table. These tables may be at any locations in micro-memory provided they can be accessed by the microprogram and cause no interference with its execution.

IV. GRAPHIC DISPLAY CONSIDERATIONS

The IBM 2250 display unit had to be the user's interface with the hypothetical computer. The programming and operation of the 2250 had to be learned from scratch and applied to the particular requirements of the hypothetical computer. During that period of learning, a number of unique problem areas appeared; these problems were not always covered in the IBM literature but by experimentation it was determined which problems could be solved and which problems had to be approached from a different direction. A great deal was learned about the actual capabilities of the 2250.

A. THE DISPLAY UNIT

The computer facility at NPGS owns one IBM Display Unit, Model 1. This display unit is connected to the school's IBM 360/67 computer through a multiplexer channel. At the time this work was done, the 2250 operated under the System/360 Operating System along with the normal flow of batch processing. The 2250 could only operate from the number 1 CPU however.

The 2250 takes the place of one of up to eight control units which may be attached to a channel. Both data to be displayed and control information necessary to direct the operation of the 2250 are transferred over the channel. The 2250 is built around a Cathode Ray Tube upon which graphic and alphanumeric information is displayed.

IBM 2250 Display Units may be configured in a variety of ways; the 2250 at NPGS has the following special features:

Absolute Vectors and Control - Allows vectors of any length and at any angle to be drawn on the CRT. Without this feature, only horizontal and vertical vectors of any length and 45° vectors of limited length may be drawn.

Character Generator - Provides a standard set of 63 alphanumeric and special characters in either basic or large size as selected by the programmer. The character generator translates one System/360 byte representation of an alphanumeric character into a sequence of analog signals necessary to trace the character on the CRT display.

Alphameric Keyboard - Provides a typewriter-like keyboard with which the user can perform editing or compose messages. When the 2250 is equipped with a character generator, messages are sent in character format to the 2250 computer.

Programmed Function Keyboard - Consists of keys, indicators, and sensing switches for use with replaceable descriptive plastic overlays. The plastic overlay is pictured and the sensing switches are described in Ref. 3. In this program the sensing switches are not used, as only one overlay is required. There are thirty-two function keys, each with a built-in indicator light. The function of each key is program-defined; in this program, the required keys are enabled by the program and identified to the user by being lighted at the appropriate times. Most times the user will have a selection of lighted keys from which to choose the particular sequence of orders he wishes executed.

Light Pen Detect - The light pen, when pointed at an illuminated portion of the screen, identifies that portion to the computer. The action taken by the computer is program-defined in much the same manner as the function keys.

Buffer Storage - The 2250 is provided with a local buffer in which to store image orders for display regeneration purposes. By the use of this buffer, the 2250 can operate concurrently with the main computer, freeing the CPU, main core, and the channel for other purposes. Buffer sizes of 4096 or 8192 bytes are available. The 2250 at NPGS is configured with the 4K buffer.

A more detailed explanation of these features plus descriptions of the Graphic Design feature (not available at NPGS) and the Operator Control Panel may be found in Ref. 3.

B. CREATING THE DISPLAY

While learning to effectively utilize the 2250 Display unit, a number of unique problem areas were encountered. Many of these problems centered around the amount of local buffer space available. The 2250 at NPGS as presently configured, has only 4096 bytes of local buffer storage. This is adequate for most programs run on the 2250, but becomes a severe limitation when large amounts of data are required to be displayed. Many problems would not have been encountered and much valuable time would not have been spent trying to correct these problems if the 8192 byte buffer had been installed.

In the initial stages of programming the 2250, the manner in which the display orders were generated in main core and the manner in which these orders were placed in the 2250 buffer was not fully evident. The primary objective at that time was to learn to program the 2250 in Fortran IV and to cause a picture to appear on the screen. The early efforts at accomplishing this task were eventually successful, but upon closer examination at a later date, the methods used were shown to be cumbersome and not very efficient.

The building of a hypothetical computer, displayed upon a screen, required first of all that a great many lines be drawn. Once the outline of the memory and register structures was in place, labels for individual boxes were required. Finally, the data was to be placed in the boxes. When the original program was run it was found that all of the data was not being displayed in the memory locations on the screen. It was assumed that the 2250 buffer had filled up before the entire presentation could be displayed.

It was found that some 2250 buffer space could be saved by more efficient coding of that portion of the program concerned with generating lines and boxes. This portion of the coding was rewritten and all of the data was then able to be displayed.

The initial concept of the hypothetical computer included a micro-programming capability, hence there was also a requirement for a display of micro-memory locations. All of the coding for the primary display was revised to eliminate all possible inefficiencies and coding was written for the basic structure of the micro-memory display. The micro-memory display was initially executed in Omit status so that it would not show up on the screen; the primary display was executed in Include status and was displayed normally. The program was written so that the use of certain Programmed Function Keys would cause the visible display to change from one picture to the other. This scheme worked but at that point no labels or data had been included in the micro-memory display and it was felt that the 2250 buffer would surely be exceeded again if this additional requirement was added to the program.

Orders used by the 2250 in generating a display are fully explained in Ref. 3. To determine just how much 2250 buffer space was being used and just how it was broken down among the various graphic data sets, it

was necessary to know what graphic orders and data were being produced. The actual graphic order consists of two consecutive bytes, the Set Mode (SM) and the Mode Control (MC) byte. The SM byte always contains a hexadecimal 2A. The MC byte contains a variable code such as 00, 40, 80, FF or others. The MC byte determines just what mode (graphic, character, etc.) is to be entered.

Following an MC byte may be a variable number of associated data bytes. The SM byte enables the 2250 circuitry to detect the presence of a new order; the next SM byte in effect resets or clears the present mode of operation and the MC byte defines the new mode of operation.

Vectors may be generated in either the absolute or incremental mode; all plotting in this program was done in the absolute mode. Four bytes are required to identify one absolute x, y coordinate on the CRT display. These bytes are arranged as shown in Figure 8.

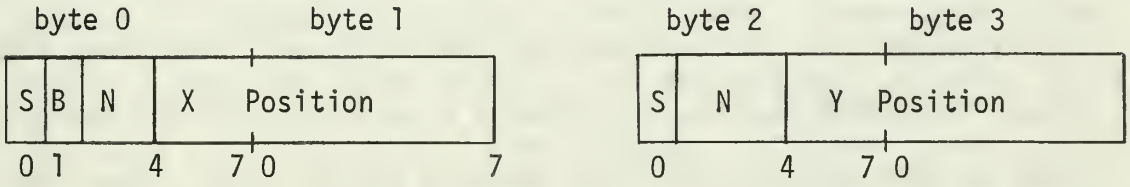

FIGURE 8. Absolute graphic coordinates.

The S bits are sign bits assigned internally; the N bits are also assigned internally to conform with the sign bit. The B bit is the blanking bit, and it controls intensification of the x, y coordinate. Some graphic orders only position the beam without intensifying it (MVPOS and STPOS). A following graphic order will usually cause the beam to be unblanked in order to draw a line or generate a character.

It was relatively easy to determine the number data bytes that would be required when generating lines on the screen. Since all of the lines

were generated in graphic data sets separate from the text, only one order would be generated. This order would be Enter Graphic Mode, Absolute Vector and would consist of the two bytes 2A 02. The remaining bytes in the graphic data set would consist of data in the four byte sequences shown above.

The character mode orders were not so easy to figure out. Reference 3 does not give a fine enough breakdown of character orders to determine as closely just how many bytes are actually used. The two order bytes are present as always and they are followed by data bytes - one byte per character. After the first look this seemed to be the extent of it and estimations were made accordingly.

To pin these figures down however, a printed output of the contents of each graphic data set within the buffer was desired. The GSPRD routine reads text information or all orders and data from part or all of a graphic data set from the 2250 buffer into a main storage array. The main storage array used was declared LOGICAL *1 and dimensioned 4096 bytes. The GSPRD routine also returns a value which is the number of bytes of data actually read into the main storage array. Once read into the main array by GSPRD, the graphic orders and data could readily be printed out, byte by byte, in hexadecimal form. Additionally, the value of the number of bytes actually read, was printed out in integer format.

The printout obtained, verified the estimates of 2250 buffer space being used by a graphic data set when that data set was generating orders in the graphic mode. These estimates were within two to four bytes of the actual figure with a common exception which is discussed in a moment.

Those graphic data sets generating orders in the character mode however, actually used a significantly greater amount of space than was

originally estimated. The printout of these graphic data sets showing exactly how character mode orders were generated proved to be of great value later on when the final display was decided upon and generated.

This printout showed that besides the two byte order setting the character mode and the appended bytes describing individual characters, there were ten additional bytes per element generated. These bytes included two order bytes and two address bytes which could cause an unconditional transfer, two order bytes setting the graphic mode, and four bytes denoting a set of x, y coordinates to set the beam position prior to beginning the generation of characters.

Individual elements were given correlation values so that each one could be identified within the graphic data set; this was in conjunction with the update facility of GSP which is described at the end of this section. The first two bytes in the sequence determine whether the element is currently in include or omit status. These bytes define either a four byte No-Op (2A C0) or an unconditional transfer (2A FF). The third and fourth bytes in the sequence are either ignored in the case of the No-Op, or they contain the buffer address of the location of the next buffer operation to be executed by the 2250 in the case of the unconditional transfer. When an element is in include status, it has the four byte No-Op; an element in omit status has the unconditional transfer. The next six bytes were required to position the CRT beam prior to the generation of each element. Since the extra ten bytes were required of each element, it followed that a few large-sized elements could effect a significant saving of 2250 buffer space over a great number of small elements.

An additional piece of information which could only be guessed about prior to this time, was brought out by the graphic data set printouts. Each graphic data set makes use of a 256 byte graphic data output area in main core. The graphic orders and data for a particular graphic data set are built up within this 256 bytes and then the entire 256 byte block is transferred to 2250 buffer storage. If a graphic data set used 1024 bytes this block would be filled and transferred four times, and so on. However, if a graphic data set uses less than 256 bytes or less than an integral multiple of 256 bytes, the entire 256 bytes or multiple of 256 bytes is transferred to the 2250 buffer. This process results in the potential wastage of one to 255 bytes of 2250 buffer storage per graphic data set. This problem area was discovered by comparing the addresses of the graphic data set printouts. In each case, if a graphic data set took less than an integral multiple of 256 bytes, there was an unused area making up the difference prior to the beginning of the next graphic data set.

Five graphic data sets in the primary display plus one in the micro-memory display were being used when the above problem was discovered. By reducing the two displays to a total of three graphic data sets, some additional space could be saved, but there would always be some portion of the 2250 buffer rendered useless. In addition to the probable non-used spaces in the 2250 buffer, the graphic subroutine package (GSP) uses 256 bytes for control purposes. It may be seen that at most, the user has only 3840 bytes of 2250 buffer to work with. In practice this figure may be reduced by anywhere from one to 255 additional bytes per graphic data set used.

Two displays were generated and executed using the include and omit features of GSP. Using this scheme, one display is generated in include status, the other in omit status; the graphic orders and data for both displays are placed in the 2250 buffer. As was discussed on page 41, the first two bytes of each element determine whether that element will be displayed or ignored at any particular time. The picture may readily be switched from one display to the other but no 2250 buffer space is saved by this method. This scheme could not have been carried out, for the amount of data required to be displayed as 2250 buffer storage was not sufficient.

The only way in which 2250 buffer space may actually be freed is through the use of the RESET subroutine. The RESET subroutine removes selected elements or all elements from a graphic data set. This routine terminates the display image produced by the concerned elements, frees all 2250 buffer storage occupied by these elements and erases all keys or correlation values associated with the removed elements. The graphic data set is not terminated by this RESET subroutine however, and new elements can be generated for the graphic data set in the normal manner.

An obvious solution to the problem of insufficient buffer storage was the use of the RESET subroutine at those times that the user desired to change the display presentation. This scheme was rejected for a number of reasons. It would necessitate the complete regeneration of a new display including all of the correct data each time the user wished to change the screen's presentation. Besides being very cumbersome and a potentially complicated and time-wasting procedure, this was deemed to be an unrealistic procedure.

The rejection of the above procedure did not completely rule out the use of the RESET subroutine however. Reseting was still the only way to completely free portions of 2250 buffer storage. The only other scheme that seemed reasonable required the use of the GSPRD and ORGEN subroutines. The former had been used previously to read the graphic orders and data of a graphic data set into a main storage array for analysis. The ORGEN subroutine moves graphic orders and data from a main storage array into a graphic data set without calling the image generation subroutines. Even though the description of the ORGEN subroutine was somewhat sketchy, it was felt that a combination of the GSPRD, RESET, and ORGEN subroutines might allow two displays without the problems of display regeneration.

The ORGEN subroutine required that all graphic orders be in the format required by the 2250, and that the orders be written as if they were at 2250 buffer address zero. Since the graphic orders in question would have been transferred from the 2250 buffer, the first requirement would be taken care of automatically, the second requirement would have to be taken care of by programming.

It was decided to test this hypothesis with the first graphic data set generated in the program. This graphic data set displayed the lines and boxes required for the primary display, and required no addresses which would need conversion; additionally there would be no updating or changing required of the elements of this graphic data set.

The program was run and the graphic data set was printed out after it was generated and each time it was read into main memory. Each time the elements of the graphic data set were read out the graphic data set was reset to free the 2250 buffer storage it had occupied. The first

time it was brought back from main storage by the ORGEN subroutine only a small portion of the elements were displayed. When the graphic data set was read out and brought back again, no part of it showed up on the display. Each printout of the graphic data set was identical however, indicating that the graphic orders and data were not being changed each time. It was thought that the graphic set was being split up and placed into various free sections of the 2250 buffer when it was brought back in by the ORGEN subroutine.

The program was run again with the various graphic data sets all generated prior to the reading out and resetting of the graphic data set in question. This procedure artificially forced the 2250 buffer space vacated by that graphic data set to remain unused until the graphic orders and data were brought back into the buffer by the ORGEN subroutine. This time the picture was normal when it was brought back into the 2250 buffer. Unfortunately this scheme didn't actually free up any 2250 buffer space and it had to be discarded.

The inability of the programmer to read out the entire contents of the 2250 buffer, and the sketchy explanations in the IBM FORTRAN manual for the 2250 display unit prevented any further work being conducted along the lines of a two picture display without an unjustifiable and potentially harmful loss of time. A single picture display incorporating all of the features of the hypothetical computer, was required.

No display scheme would be able to show all of the memory and register contents of the hypothetical computer at one time. The main memory is configured as 1000 words, the micro-memory as 999 words and there are 100 general purpose registers provided. Since any one user's program would probably only require a portion of these various storage units,

only those which were most directly affected by the program would need to be displayed. It was decided to allow the user to define his own display so that he could show any combination of memory and register locations he desired, in any order he desired, within the basic display limitations.

The basic display is configured as four columns of twenty-five boxes each. The labeling and contents of these 100 boxes are defined by the user. There are three additional 60-character boxes at the bottom of the screen labeled INPUT, OUTPUT, and KEYBOARD. The INPUT box displays the next card image to be read by the user's program; the OUTPUT box displays the results of an output or write operation within the user's program or diagnostic messages. The KEYBOARD box has a cursor inserted into it by the master program and is used to input changes to the display or the user's program via the alphameric keyboard.

Two graphic data sets are utilized in generating the display. In this way, wasted space within the 2250 buffer is kept to a minimum. The lines and boxes of the basic display are generated as elements of a single graphic data set. These elements actually use 950 bytes of 2250 buffer storage; since the buffer is filled in 256 byte increments however, 1024 bytes are actually required.

The remainder of the picture, which includes all alphanumeric data displayed on the screen, is generated in the second graphic data set.

Ideally the data required for each of the 103 boxes on the screen would be generated as individual elements. After having determined how the graphic orders were actually generated in the character mode however, it was easy to determine that 103 separate elements might require excessive 2250 buffer storage. Specifically, each of the first 100

elements would require a minimum of fourteen data bytes (five for the label, one for a blank, and eight for the contents) plus two graphic order bytes plus the additional ten transfer and positioning bytes, for a total of twenty-six bytes per element. The three lower elements would require seventy-two data bytes plus the twelve additional bytes each. The total number of bytes required would be 2852; the next higher multiple of 256 is 3072 bytes which would be the actual amount of 2250 buffer space taken up. Since the first graphic data set required 1024 bytes and GSP required 256 bytes, the overall total would come to 4352 bytes which would exceed the 2250 buffer.

To cut down on the amount of 2250 buffer space used, the number of elements generated had to be kept to a minimum. In addition, the contents and labels of each of the 100 display boxes and the three special purpose boxes had to retain the capability to be updated, substituted for, or removed, as individual units.

The four columns of boxes are placed at equal distances apart so that nineteen characters fit between the leading edge of one box and the leading edge of the next box to the right. This leaves space for seventy-two characters across the display with the fifth, twenty-fourth, forty-third, and sixty-second characters falling immediately to the left of the left edge of each of the four boxes in each row. There are five character positions to the left of the first box, eleven character positions within each box, and eight character positions between the boxes.

The first five character positions of each nineteen character segment are used for the box labels, the next character position (inside the left of the box) is a blank, and the next eight character positions comprise

the memory word. The remaining character positions were blanks up to the beginning of the next label. No matter what type of label or what the contents of the memory word, the character positions for each nineteen character segment and for each of the first 25 rows of seventy-two characters, were all consistent.

The bottom three rows were labeled INPUT, OUTPUT, and KEYBOARD followed by blanks out to a total of 72 character positions. Within the boxes a maximum of 60 characters could be shown.

The arrangement of characters upon the screen as described above led to a relatively simple method of generating the associated graphic data set. Each of the 28 rows of 72 characters was generated as an element of the graphic data set. Under this arrangement each element would require the 12 graphic order, transfer, and positioning bytes plus 72 data bytes, for a total of 84 bytes per element. The 28 elements of the graphic data set were estimated to require a total of 2352 bytes which would actually require 2560 bytes of 2250 buffer space. The overall total was then estimated to be 3840 bytes, well within the 2250 buffer's capabilities.

An array called DSPLY declared as LOGICAL *1 and dimensioned 72 x 28 was decided upon as the basis for the display. To begin with, the entire array was set to blanks; those character positions which would not be used throughout the running of the program always remained blank, insuring the consistency of the display.

C. THE GRAPHICS UPDATE FACILITY

This program makes a great deal of use of the update facility of GSP. The update facility allows individual elements of a graphic data set to be modified without affecting any of the other elements of the graphic

data set. The only requirement is that the replacement element be generated by the same subroutine that created the original element, and that the element be keyed or correlated for identification. Program assigned correlation values of one to twenty-eight are used for each of the elements concerned. The update facility of GSP is described in detail in Ref. 1.

The subroutine used to generate the 28 elements was the PTEXT subroutine. This subroutine creates an element that displays up to 74 characters using the character generator feature of the 2250. The characters are displayed in the same format as they appear in the DSPLY array in main storage. Each of the 28 elements created by calls to PTEXT may be modified by further calls to PTEXT utilizing the update facility of GSP. Each element is 72 characters in length and each update call regenerates all 72 characters; only those characters actually changed within the DSPLY array however actually show up as a change on the screen. While an entire row is being regenerated, it looks as though only those segments of an element actually modified are changing.

The PTEXT subroutine used to do all of the updating of the first 25 elements is itself part of a program subroutine called UPDATE. The UPDATE subroutine causes the correct portion of the DSPLY array to be modified and passes the appropriate DSPLY element, correlation value, and set of X - Y coordinates to PTEXT for updating.

While enabling the system to operate efficiently, the update facility also allows the hypothetical computer to operate in a fairly realistic manner. The hypothetical computer gives an educational and surprisingly realistic look at the insides of an operating computer to the user.

V. OPERATING INSTRUCTIONS

The operating instructions cover all of the specific areas that the user must be familiar with in order to operate the hypothetical computer. The user does not have to refer to any IBM publications to operate the 2250 display unit. The 2250 is set up as the operator's console of the hypothetical computer. The functions of the alphameric keyboard, the Programmed Function Keys, and the display itself are explained as they apply to the operation of the hypothetical computer. Examples of the functions of the various function keys and control cards are given followed by short discussions on efficient use of CPU time and the problem of display flicker. Instructions on preparation and submission of the card deck for a run are included.

A. PROGRAMMED FUNCTION KEYS

There are 32 programmed function keys associated with the 2250 display unit. Any number of these keys may be enabled by the programmer to be used by him as he sees fit. Each key also has a light inside it, and these lights are also under control of the programmer. The lights are primarily for the convenience of the user to let him know which PF keys are currently applicable to the correct running of the program. Other attention sources include the END key of the alphameric keyboard and the light pen. A detailed general description of attentions and the attention handling subroutines of GSP is given in the section entitled "Communicating With The 2250 Operator" in Ref. 1.

All of the Programmed Function keys described below with the exception of P. F. key number 4 are lighted concurrently. This does not mean that all of them will produce meaningful results at all times, in a few particular instances some of them will produce disastrous results. The user is advised to become familiar with these P. F. keys so that he always knows what to expect when he depresses one of them.

Prior to any P. F. keys being lighted a number of steps are taken by the master program.

1. The arrays which will hold the contents of the micro-memory and main memory words and the contents of the registers are initialized to zero.
2. The Graphic Subroutine Package (GSP) is initialized and linked to the master program and the two graphic data sets are initialized. The 2250's audio alarm is sounded to notify the user that the hypothetical computer is being initialized.
3. The orders are generated within the first graphic data set to draw the four columns of 25 boxes each and the three special boxes at the bottom of the screen.
4. The entire DSPLY array is initialized to blanks and the orders and data are generated for the second graphic data set. These are the 28 elements of 72 characters each. The first 25 elements are initially composed of 72 blanks each. The last three elements are initialized with the words INPUT, OUTPUT and KEYBOARD in the leftmost positions.
5. The first of the user's control cards is read in and displayed in the INPUT box. The master program always stays one card ahead of the user's program; the next card the user will read, whether it is a control card or a data card, is always displayed in the INPUT box.

Following these initial steps P. F. keys 1, 2, 3, 5, 7, 9, 22, 25, 27, and 30 are lighted and the machine is placed in the wait state, awaiting an attention from one of the lighted keys. These P. F. keys give the user positive control over all aspects of execution of his program. Since he has this positive control, however, it is incumbent upon the user to fully understand what he is doing at all times. Following is a detailed description of the functions of these keys.

1. Process Control Card (P. F. key 1)

This P. F. key causes the user's next control card to be processed. The card to be processed is displayed in the INPUT box. Since the use of P. F. key one is directly concerned with how control cards are processed, the discussion of control cards following this section must be completely understood.

2. Read The Next Card (P. F. key 2)

This P. F. key causes the next card in the user's sequence to be read and displayed in the INPUT box. The card image currently displayed in the INPUT box is ignored. Although the user may have information in all 80 columns of the card, only the first 61 columns will be displayed due to space limitations on the screen. During the processing of control cards or during the actual execution of the user's program, the next card will usually be read automatically and the user will not need to press key number two. If the user desires to flush out the remainder of his program however, in order to begin a second program, he may use the READ key to read and ignore as many cards as necessary until the next appropriate control or other card appears in the INPUT box. After the card has been read, the machine returns to the wait state.

3. Go (P. F. key 3)

This P. F. key places the hypothetical computer into the go or run mode. In this mode instructions are executed automatically in sequential fashion. In addition, P. F. key number four, the STOP key, is lighted and remains so while the machine is computing. If not stopped by the user, the computer will execute his entire program and then place itself in the wait state. The user may then terminate, or reset and execute another program. At any point during execution in the GO mode the user may stop execution by means of the STOP key.

4. Stop (P. F. key 4)

While executing in either the GO mode or the SINGLE INSTRUCTION mode, the computer checks for an attention from P. F. key number 4 at the completion of each micro-instruction cycle. If an attention has occurred, execution is stopped and the machine places itself in the wait state. The machine generally operates at a relatively slow speed - the user can see the micro-instruction location counter changing. Usually however, the user will not be able to stop execution at a particular micro-instruction. With luck he should be able to stop within two micro-instructions of any given instruction.

5. Single Cycle (P. F. key 5)

The P. F. key causes the execution of one micro-instruction, after which the machine is again placed in the wait state. Each minute action of the user's program may be followed in detail by operating in this mode. Since there may be as many as 30 or 40 micro-instructions per machine instruction in a program this mode of operation is very slow. It would not be advisable to attempt to execute an entire program in this mode alone unless that program were very short.

6. Single Instruction (P. F. key 7)

This P. F. key causes the execution of one machine instruction, after which the machine is again placed in the wait state. While the micro-instructions making up the machine instruction are being executed, P. F. key number four, the STOP key is lighted. Execution may be stopped at any time prior to the normal end of the machine instruction cycle by depressing the STOP key.

7. Reset (P. F. key 9)

This P. F. key resets micro-memory locations one through ten to zeroes and resets micro-memory location 11 to 201. The Reset key must always be pressed to set the hypothetical computer up prior to beginning execution of a new program or prior to beginning execution of a program over again. If two sets of data have been submitted for a single user program, the Reset key may be used in conjunction with the Read key (#2), to discontinue execution of the program prior to attaining a normal ending with the first set of data. The Read key can be used to flush the remaining data cards in the first set and display the first data card of the second set in the INPUT box. The Reset key will then set the hypothetical computer up to rerun the user's program using his second set of data cards. If the user fails to provide data cards for a Reset program, the first read instruction in his program will attempt to read an end-of-file card or other control card. If this happens, the message END OF DATA will appear in the OUTPUT box and the user will have to take appropriate action. If the user's program attempts to read beyond all of his data and control cards, the message END OF FILE will appear in the OUTPUT box. If an attempt is made to read past the end of file a second time, System/360 will terminate the run abnormally.

8. Hard Copy Printout (P. F. key 22)

This P. F. key causes a printout of the DSPLY array. The printout will be a part of the normal printed output received after the termination of the run. The printout will exactly duplicate the picture on the screen with the exception of the outlines of the boxes. This feature cannot be used indiscriminately, since only one printout appears per page and if too many pages were printed the available SYSOUT space could be exceeded. About fifteen separate printouts may safely be executed. Examples of the hard copy printout are shown on pages 87 and 88 .

9. Terminate (P. F. key 30)

This P. F. key terminates the master program and erases the picture from the 2250 display unit. This is the normal manner in which a program run is terminated on the hypothetical computer. This key will terminate the run at any time so it must not be pushed until the user has completed his work.

P. F. keys numbers 25 and 27 will be described in connection with the use of the alphameric keyboard.

B. ALPHAMERIC KEYBOARD

The alphameric keyboard is used to perform two basic tasks. One task involves changing the contents of a memory word or register which is currently being displayed; the second task involves changing which memory locations or registers are to appear on the display.

Characters may be inserted onto the display screen only where the cursor appears. This system automatically inserts the cursor into the leftmost portion of the KEYBOARD box at the time the program and display are being initialized. Up to 60 characters may be inserted in a left to

right sequence in this box. In actual usage, the maximum number of characters the user would be required to insert is fourteen. As an alphameric key is pressed, the character appears on the screen immediately above the cursor and the cursor moves to the right one character position. When the information has been entered, P. F. key 25 or 27 is pushed depending upon the data entered. These P. F. keys cause routines to be entered which read the KEYBOARD data, interpret it, and change the display accordingly. The last character string inserted will remain displayed until new characters are inserted over them by the user.

When the user desires to insert new data into the KEYBOARD box he must backspace the cursor to its original position prior to typing in the new information. The old characters on the screen need not be erased prior to entering new characters; the new characters will automatically replace the old ones. If there are not sufficient new characters inserted to completely eliminate the old ones, the user should advance the cursor by means of the space bar until all of the remaining old characters have been erased. The user may then push the proper P. F. key to read and execute the data.

The 100 display boxes may show any combination of memory words, micro-memory words, or registers that the user desires. The initial display is normally set up by $DISPLAY control cards. The boxes are numbered from 1 to 100 with boxes numbered 1, 26, 51, and 76 heading each of the four columns. Once the user's program and the display have been initialized, the user is free to change his display as often as conditions warrant.

1. Changing The Contents of a Word or Register

A change in the contents of a memory word or register requires the user to give the type of location (L - micro-memory, M - memory, R - register), the location number, and the replacement data. This information may be inserted in any one of a number of ways. If the display box number 50 looks like this,

R-092 00000021

and the user desires to change the contents of R-092 to 44, the user may accomplish this change by inserting into the KEYBOARD box one of the following:

R-092 00000044 or R92 44 or R92,44 or R92,00000044.

There must be a blank or a comma between the label portion and the contents portion of the input string. The first method shown is recommended for the new user as this exactly duplicates the format of the display and mistakes would be minimized; all methods are equally valid however.

Once the string has been inserted into the KEYBOARD box the user must depress P. F. key number 25. This attention causes the newly entered string to be read and the contents of R-092 to be updated accordingly. The appropriate portion of the display will visually update almost immediately.

Further examples:

a) L-050 01200040 to b) L-050 01200042

Insert from the keyboard either the string b, or,

L50, 1200042 or L50 01200042

or either of the above with a blank substituted for a comma or vice versa. When any change of contents is made, the first non-zero digit in the

contents portion and all following digits must be entered from the keyboard. If fewer than eight digits are entered for the contents portion, the system will in all cases assume leading zeroes.

The content change feature of the alphameric keyboard may also be used by the programmer to input his program. The capability exists to input the entire micro-program and machine program. In practice this would be extremely cumbersome as these two sections might total 150 to 200 separate instructions. It would certainly be feasible to manually input the machine program however, especially if it were short (15 instructions or fewer).

2. Changing the Locations Being Displayed

Two different but interrelated functions come under this heading. One or more memory locations or registers may be inserted into display boxes whether those boxes are currently empty or contain other memory locations or registers, or currently displayed memory locations or registers may be erased from the screen and be replaced by blanks. The format of both functions is similar.

If the user wishes to insert micro-memory locations 101 through 120 into boxes 26 through 46 he would insert the following into the KEYBOARD box:

L101-120,26 or L101-120 26

These two formats are the only ones accepted by the system. The required information includes L, M, or R, the range of locations or registers to be displayed, and the box number in which the sequence is to begin. If only micro-memory location 101 were to be inserted into box number 26, the user would insert the following into the KEYBOARD box:

L101,26 or L101 26

When the string is complete, the user must depress P. F. key number 27. If boxes 26 through 46 were blank, the new information would merely be inserted into the proper boxes. If any or all of boxes 26 through 46 displayed other registers or memory locations these would be removed from the display and replaced by the new information. The actual contents of the replaced registers or memory locations would not be altered in any way.

If the user wishes to erase one or more memory locations or registers from the display, the procedure is exactly as above except that zero is inserted as the box number.

M0-10,0 erases memory locations 0 through 10 from the display

R15 0 erases register 15 from the display

In all examples blanks and commas are interchangeable. As in the previous example, the actual contents of the erased memory locations or registers are not altered.

C. CONTROL CARDS

This discussion of control cards is closely allied with the use of Programmed Function Key one since this P. F. key causes the next control card to be processed. All control cards require a dollar sign ($) in column one. When P. F. key number one is depressed the card image displayed in the INPUT box is read and the appropriate processing is done; when processing is completed the next user card is automatically read into the INPUT box. In all cases when normal processing has taken place, the user sees the next card to be read and used by the program displayed in the INPUT box. Since he can see what the card looks like the user probably won't attempt to process a non-control card as a control card. If the user errs however, and the card he attempts to process does

not have a dollar sign in column one, the hypothetical computer will display the diagnostic message NOT A CONTROL CARD in the OUTPUT box and the system will return to the wait state. If the card has a dollar sign in column one but beyond that does not fit the description of any control card, the message UNRECOGNIZABLE CNTL CARD will appear in the OUTPUT box, and the hypothetical computer will return to the wait state. In either case, when a control associated diagnostic message appears, the next card in the input stream is not read or displayed. The card image which is causing the diagnostic message to appear remains available in the INPUT box until the user does something other than attempt to process it as a control card.

A control card may be one of four basic types: $COMPILE, $READ, $DISPLAY, or $PRINT. The end of file card has only a $ in column one and is blank otherwise; it may also be thought of as a control card. On the four basic control cards there may be one or more blanks between the $ and the control card definition word.

The user has the option of inputing his micro-program in a higher level source code which will then be compiled by the master program into the correct object code, or he may input his micro-program directly in object code.

The $COMPILE control card is required if the user wishes to input his micro-program in a higher level source language. This control card initiates the compiler which takes over and operates on the user's source coded cards which must immediately follow the $COMPILE card. Following $COMPILE on the same card the user is allowed two optional parameters. The word SOURCE will cause a printout of each source coded statement; the word OBJECT will cause a printout of each appropriate micro-memory

location and the object code going into that location. If both parameters are included, the printout of each source code statement is immediately followed by the object code generated in the appropriate micro-memory locations. An example of this output is as follows:

```
Let     L2 - M (L1)  Higher level language statement
       201    10019
       202   180024  Micro-memory locations and object code
       203   250002

Let     L1 - L1 + 1
       204    10024
       205  8990025
       206   250001

Put    (181, 188,L2)
       207  8980022
       208  8970023
       209    20021
            .
            .
            .
```

The $READ control card is required to read in an object coded user's micro-program. The format of the $READ card is as follows:

$READ ···L201-350

In the example, the $READ card will cause data to be read into micro-memory locations 201-350. The data (in this case the user's micro-program in object form) must appear right justified in eight fields of ten digits each per card (8I10 format). The $READ card is also used to read in the user's machine language program; in this case the letter M followed by the range of locations would replace L201-350 in the above example. The user's machine program must appear in the same format (8I10) as the micro-program and must follow the $READ control card. The $READ card will also read information in 8I10 format into a series of registers by using R instead of L or M. In actual practice the latter would be used little if at all.

The cards required to read in a users machine language program of four instructions into words 0-3 would appear as follows:

```
$READ ··· M0-3
01000020 03100020 04100022 0200022
```

The $DISPLAY control card tells the hypothetical computer which memory or micro-memory locations or which registers to display. These control cards are usually used to set up the initial display. The $DISPLAY card is not required however, since the user may define or redefine his display at any time through the keyboard input feature. The format of the $DISPLAY card is shown in the following examples.

Examples

$DISPLAY ··· L1-25,1	displays micro-memory locations 1-25 beginning in box number one.
$DISPLAY ··· R1-8,26	displays registers 1-8 beginning in box number 26.
$DISPLAY ··· M0-15,0	erases the display of memory locations 0-15.

In all cases there must be at least one blank between DISPLAY and the L, R, or M. The format of the data portion of the $DISPLAY control card is identical to the format required when inputing similar information via the alphameric keyboard.

The $PRINT control card causes the contents of a series of memory, micro-memory, or register locations to be printed out under an appropriate heading. The format of this control card is shown in the example.

The hard copy printout produced by the $PRINT card is somewhat analogous to a System/360 dump. Only the first location number and every tenth location number following in a sequence is printed out on the left side of the page. Across the page are printed the contents of the printed location number and the following nine locations. The control card must show the type of information (L, M or R) to be printed and the range.

$PRINT ··· L201-350

causes a printout in the following general format.

MICRO-MEMORY

```
ADDR
 201   00000000  00000000  00000000  . . .  00000000
 211   00000000      .
 221      .          .
  .       .          .
  .       .
  .
```

The printouts of M and R locations are similar.

D. CONTROL CARD EXAMPLES

Following are some examples of typical input sequences with explanations of what functions are performed by the control cards.

Example 1

```
$COMPILE  SOURCE  OBJECT
User's source coded microprogram (including an END card)
$DISPLAY L201-300,1
$PRINT   L201-300
$READ    M0-20
User's machine program (3 cards in 8I10 format)
$PRINT   M0-20
$DISPLAY L201-300,0
$DISPLAY L1-40,1
$DISPLAY R1-10,41
$DISPLAY M0-24,76
User's program data
$
```

The user's micro-program will be compiled from his source code input. Additionally, printouts of each higher level language statement followed by the object code produced will be given.

Micro-memory locations 201-300 will be displayed and a printout of the contents of these same locations will be given.

The user's machine program will be read into memory locations zero through twenty and a printout of the contents of these locations will be given.

The display of micro-memory locations 201-300 will be removed from the screen and micro-memory locations 1-40, registers 1-10, and main memory locations 0-24 will be displayed.

The Reset key (#9) may be pressed and the user's program may be run, the final control cards will signal the end of his program data. Notice the similarity in the formats of the $DISPLAY, the $READ and the $PRINT control cards.

Example 2

```
$DISPLAY  L201-300,1
$READ     L201-350
Portion of user's micro-program in object code
$READ     L900-950
Remainder of user's micro-program in object code
$DISPLAY  M0-24,76
$READ     M0-14
User's machine program (2 cards in 8I10 format)
User's program data
$
```

In this example, the user's micro-program has been written in object code and the compiler is not used at all. The $DISPLAY cards function in the normal manner as in the previous example. The $READ cards also function in the same manner but in this example the user's micro-program is read in object code and furthermore, the entire program is not read into contiguous locations. The user must insure that the proper program cards follow each particular $READ control card. There are no control cards following the user's machine program so the data follows in turn. The number of end of file cards ($ ...) following the data cards is optional; there must be at least one.

Example 3

```
$COMPILE
User's source coded micro-program (including an END card)
User's program data
$
```

In this example the user's higher level language micro-program is compiled but no printouts are given. The user must input his own display setup via the alphameric keyboard. The user must also input his entire machine language program from the alphameric keyboard. The user does not have to display the locations into which he is inputing his machine instructions. If he does not display these locations he is merely inserting his data into the internal M array. In most cases however, the user would display the proper locations so that he could visually check the accuracy of the input data. In the latter case, the user would be updating the DSPLY array as well as inserting data into the M array. Frequent use of the $DISPLAY and $PRINT cards is recommended. If the user knows ahead of time just what he wants displayed, the appropriate insertions of $DISPLAY cards will reduce the number of times the relatively slow alphameric

keyboard must be used. The $PRINT card along with the printout options of the $COMPILE card and the printout feature available on P. F. key 22 give the user later access to virtually everything that goes on in the hypothetical computer. The printout features comprise a powerful debugging tool.

E. TIME CONSIDERATIONS

The hypothetical computer provides the user with two minutes of System/360 CPU time for the GO step of the run. Under normal circumstances this amount of time is enough, but under some conditions abnormal termination of the run could occur due to reaching the two minute time limit.

The 2250 display unit can place a great many more demands upon the System/360 CPU during the execution of the user's program. While the actual computation time required by the user's program may be quite modest, the time required to transfer intermediate results to the 2250 and display the information on the screen may be excessive.

The user has the capability to display any memory locations or registers he desires. The contents of some micro-memory locations (L2-L25 especially) change quite often, some will change during each micro-instruction cycle. By displaying these locations the user can really get the feel of seeing the computer working. The user must fully realize however, that he is paying a high price in CPU time to see the very great amount of updating that is going on. When running a program that requires a fairly large amount of compute time the user would be wise to not display those locations which require frequent updating. The same general rule holds if more than one user program is to be run during one session. Due to complexity or number of programs being run,

the user may, in some cases, require more than the standard allotment of System/360 CPU time. If this situation occurs, the user must make special arrangements with the NPGS Computer Facility Operations Manager.

F. FLICKER

The display of the hypothetical computer will flicker when there is a large amount of information presented on the screen. The 2250 has a variable regeneration rate up to a maximum of 40 cps; a steady display image can be obtained when the regeneration rate is greater than 30 cps. The actual regeneration rate of the hypothetical computer is not known but if all of the display boxes are full of characters the regeneration rate is somewhat less than 30 cps and the flicker is a problem. The user can cut down on the amount of flicker if he only displays those memory locations and registers which he actually requires. Otherwise flicker is a function of the 2250 display unit and is out of the hands of the user.

G. SUBMITTING A RUN

The user's control cards, micro-program, machine program and data are all put together in the order shown in the control card examples on pages 63-65 . The program consists of a FORTRAN main routine and several subroutines, preceeded by the normal job and control cards. (The phrase TYPERUN = HOLD must be on the Job card.) The GSP sub-routines reside permanently on the system disk and are linked to as described in Ref. 1. The user's deck is data for the FORTRAN program and goes between a //GO.SYSIN DD * Card and an orange /* card at the end of the program deck. The entire deck is submitted under a job submission card filled out as follows:

1. Check "SPECIAL" under Type of Run and write in "2250" in large numbers.
2. Check the boxes labeled "TESTING", "FORTRAN IV", and "OS/360 SYSTEM".
3. Write in the time of desired run in the lower third of the card. (Check current computer facility instructions for available 2250 run times.)
4. Submit at least one half hour prior to the desired run time.

VI. CONCLUDING REMARKS

The simulated microprogrammed computer described in this report gives the user the means with which he can design his own special purpose machine. The manner in which a higher-level language microprogram is used to define a machine is straightforward and easy to use. By a simple read-in, the user can alter or completely change the design of his machine.

The simulated microprogrammed computer used in conjunction with the IBM 2250 Display Console serves to give the user the feeling that he has a small computer at his own disposal. Certain of the function keys are operated as hypothetical computer controls. The user can sit at the display console and view the contents of any word or register in the machine. He also has the capability to change the contents of any word or register by utilizing the function keys and keyboard. These features give him complete control over the system, allowing him to change data, machine language instructions, and even micro-orders at any point during execution.

A number of improvements could be made to the system developed here to further extend its overall capability. For instance, it could be modified to allow variable word length machines to be simulated. A clock might be added to display the elapsed time at any point during execution. Additional "hardware" features could be implemented to include shifting, AND ing, OR ing, etc. The higher-level language might be extended for increased flexibility in writing the microprogram. A micro-assembly language could be included in the system to allow the microprogram to be

written with equivalent efficiency to an object-coded micro-program, but by using symbols and numbers instead of numbers alone.

The light pen of the graphic display could be used to mark locations being displayed for insertion or changing of data or labels. A more flexible display format might be implemented to allow the user to tailor it more to his needs. A capability to allow the user to insert and display separate statements of the higher-level language and even control cards from the keyboard could be added. For educational or instructional purposes, it might be a good idea to make movies or videotape of the display while the machine is running, to furnish a dynamic classroom demonstration of computer operation.

APPENDIX A

The hypothetical machine described in this example is based on the "stack" concept, i.e. all operations are performed on elements which are on a pushdown stack. The registers R1-R99 are used as the stack, and when the computer is running, micro-memory location L7 points to the current top of the stack. (During the execution of some instructions L6 points to the next-to-the-top of the stack.)

A word in this machine consists of eight digits. For an instruction word these eight digits are considered to be split into two four-digit groups called syllables. Each syllable is further split into a one-digit code and a three-digit modifier. A program for this computer is a string of syllables which indicate, in the appropriate order, the changes-of-state which the machine must undergo to accomplish the desired computing task. (A syllable is roughly equivalent to a "machine instruction" in the typical Von Neumann computer.)

The four valid values for the code of a syllable and the meaning of each are as follows:

0 Treat this syllable as a "no-op".

1 Load the value of the modifier onto the stack.

2 Load the contents of the memory location whose address is given by the modifier onto the stack.

3 Treat the modifier as an operation designator and perform the designated operation.

The eleven operation designators implemented for this example and the meaning of each are as follows:

1 (Read) Take the value off the top of the stack and consider it to be the number of items to be read from a data card. Then read a card into the I/O buffer. Store the value from the first ten-digit field of the I/O buffer into the memory location whose address is currently at the top of the stack. Then remove the address from the stack. Repeat the storing operation from successive fields until the specified number of items (max.8) have been read from the card. (A corresponding number of addresses will have been removed from the stack.)

2 (Print) This operation is performed analogously to the Read operation, and results in the printing of a line of up to eight ten-digit fields.

3 (Store) Store the value from the top of the stack in the memory location whose address is given in the next-to-the-top of the stack. Then remove both the value and the address from the stack.

6 (Copy) Make a copy of the value at the top of the stack and place it on the stack. (This results in the top of the stack and the next-to-the-top of the stack being identical.)

7 (Swap) Swap the values of the top of the stack and the next-to-the-top of the stack with one another.

8 (Erase) Remove the value from the top of the stack and discard it.

11 (Branch) Take the number off the top of the stack and consider it to be the address of the memory location where the next syllable to be "executed" can be found. (The first of the two syllables at that location is the one being "branched to".)

12 (Branch-on-zero) If the value at the next-to-the-top of the stack is zero, take the number off the top of the stack and consider it to be the address of the memory location where the next syllable to be executed can be found. Otherwise, simply remove the value from the top of the stack, and go on to the next syllable in sequence. (Note that in either case, what was next-to-the-top before this operation is at the top after the operation.)

22 (Subtract) Subtract the value at the top of the stack from the value next-to-the-top, and then remove the value from the top. (This leaves the result as the new top.)

23 (Multiply) This operation is similar to the subtract operation.

99 (Stop) Stop the running of the machine until a start key is depressed.

When this computer is running, location L1 is used as a location counter, L2 holds the current instruction word, L3 indicates which syllable of the word is under consideration, L4 holds the code of that syllable, and L5 holds its modifier.

The higher-level language microprogram for the machine of this example is the following:

```
IF (L3) 2, 2, 9                    (1)
LET L3 = 1                         (2)
LET L2 = M(L1)                     (3)
LET L1 = L1 + 1                    (4)
PUT (181,188,L2)                   (5)
GET (181,181,L4)                   (6)
GET (182,184,L5)                   (7)
GO TO 12                           (8)
LET L3 = 0                         (9)
```

```
      GET (185,185,L4)                    (10)
      GET (186,188,L5)                    (11)
      IF (L4) 1,1,13                      (12)
      LET L4 = L4-2                       (13)
      IF (L4) 15,18,21                    (14)
      LET L7 = L7 + 1                     (15)
      LET R(L7) = L5                      (16)
      GO TO 1                             (17)
      LET L7 = L7 + 1                     (18)
      LET R(L7) = M(L5)                   (19)
      GO TO 1                             (20)
      LET L11 = 900 + L5                  (21)

1     LET L9 = R(L7)                       (1)
      LET R(L7) = 0                        (2)
      LET L7 = L7 - 1                      (3)
      READ                                 (4)
      LET L51 = 101                        (5)
      LET L52 = 110                        (6)
      IF (L9) 16,16,8                      (7)
      LET L10 = R(L7)                      (8)
      LET R(L7) = 0                        (9)
      LET L7 = L7 - 1                     (10)
      GET (L51,L52,M(L10))                (11)
      LET L51 = L51 + 10                  (12)
      LET L52 = L52 + 10                  (13)
      LET L9 = L9 - 1                     (14)
      GO TO 7                             (15)
      RETURN

2     LET L9 = R(L7)                       (1)
      LET R(L7) = 0                        (2)
      LET L7 = L7 - 1                      (3)
      LET L51 = 101                        (4)
      LET L52 = 110                        (5)
      IF (L9) 15,15,7                      (6)
```

```
      LET L10 = R(L7)                    (7)
      LET R(L7) = 0                      (8)
      LET L7 = L7 - 1                    (9)
      PUT (L51,L52,M(L10))               (10)
      LET L51 = L51 + 10                 (11)
      LET L52 = L52 + 10                 (12)
      LET L9 = L9 - 1                    (13)
      GO TO 6                            (14)
      WRITE                              (15)
      RETURN                             (16)

3     LET L6 = L7 - 1                    (1)
      LET L9 = R(L6)                     (2)
      LET M(L9) = R(L7)                  (3)
      LET R(L7) = 0                      (4)
      LET R(L6) = 0                      (5)
      LET L7 = L7 - 2                    (6)
      RETURN                             (7)

6     LET L6 = L7                        (1)
      LET L7 = L7 + 1                    (2)
      LET R(L7) = R(L6)                  (3)
      RETURN                             (4)

7     LET L6 = L7 - 1                    (1)
      LET L9 = R(L7)                     (2)
      LET R(L7) = R(L6)                  (3)
      LET R(L6) = L9                     (4)
      RETURN                             (5)

8     LET R(L7) = 0                      (1)
      LET L7 = L7 - 1                    (2)
      RETURN                             (3)

11    LET L3 = 0                         (1)
      LET L1 = R(L7)                     (2)
```

```
      LET R(L7) = 0                          (3)
      LET L7 = L7 - 1                        (4)
      RETURN                                 (5)

12    LET L6 = L7 - 1                        (1)
      IF (R(L6)) 5,3,5                       (2)
      LET L3 = 0                             (3)
      LET L1 = R(L7)                         (4)
      LET R(L7) = 0                          (5)
      LET L7 = L7 - 1                        (6)
      RETURN                                 (7)

22    LET L6 = L7 - 1                        (1)
      LET R(L6) = R(L6) - R(L7)              (2)
      LET R(L7) = 0                          (3)
      LET L7 = L7 - 1                        (4)
      RETURN                                 (5)

23    LET L6 = L7 - 1                        (1)
      LET R(L6) = R(L6) * R(L7)              (2)
      LET R(L7) = 0                          (3)
      LET L7 = L7 - 1                        (4)
      RETURN                                 (5)

99    STOP
      END
```

The example program written for this computer reads a number (N), computes its factorial (NF) by calling a recursive function, prints the values of N and NF, and then goes back to read another card. The factorial function returns a 1 when its argument (M) equals zero; otherwise, it calls itself to evaluate (M-1)!, and then it returns the value so obtained, multiplied by M.

This example program would be roughly the following in FORTRAN (Of course, the recursive function is invalid in that language.):

```
    1 READ (5,2) N
    2 FORMAT (I10)
      NF = NFACT(N)
      WRITE (6,5) N,NF
    5 FORMAT (2I10)
      GO TO 1
      END

      FUNCTION NFACT (M)
      IF (M.EQ.0) GØ TØ 15
      NFACT = M*NFACT(M-1)
      RETURN
   15 NFACT = 1
      RETURN
      END
```

The machine language version of this example program appears on the left in the following table, and a pseudo-assembly language version appears on the right:

LOC	SYL1	SYL2	LOC	SYL1	SYL2
0	0000	0000		No-op	No-op
1	1021	1001	BEGIN	N	1
2	3001	1023		Read	NF
3	1005	2021		BACK	(N)
4	1011	3011		FACT	Branch
5	3033	1023	BACK	Store	NF
6	1021	1002		N	2
7	3002	1001		Print	BEGIN
8	3011	0000		Branch	
9					
10					
11	1018	3012	FACT	RET1	Br-on-z
12	3006	1016		Copy	RET2
13	3007	1001		Swap	1
14	3022	1011		Subtr	FACT
15	3011	0000		Branch	
16	3023	3007	RET2	Mult	Swap
17	3011	0000		Branch	
18	3008	1001	RET1	Erase	1
19	3007	3011		Swap	Branch
21			N		
23			NF		

APPENDIX B

In this example a simple BRAIN-2 machine is defined. Each word can hold an unsigned integer constant, four digits in length. The machine language instruction contains two parts, an opcode part one digit long and an address part three digits long. During execution, L3 is used to hold the opcode part and L4 is used to hold the address part. Also, L1 is used for the location counter and L2 is used as the instruction register.

The machine uses an accumulator (a four digit register) in which all arithmetic operations are performed. If overflow occurs, (e.g. > 9999), the high order digits are discarded. If underflow occurs (e.g. < 0), the contents of the accumulator are set to zero. Also, a working register is used for intermediate storage. During execution L6 is used as the accumulator and L7 is used as the working register.

The operation codes used and their corresponding meanings are shown below.

0 HALT stop processing this job

1 READ a four digit number from a data card (columns 1-4)

2 LOAD the contents of the specified address into the accumulator

3 STORE the contents of the accumulator at the address specified

4 ADD the contents of the specified address to the value in the accumulator

5 SUBTRACT the contents of the specified address from the contents of the accumulator

6 CIRCULAR LEFT SHIFT the contents of the accumulator 0, 1, 2, or 3 places depending upon the contents of the address portion of the instruction

7 RIGHT SHIFT the contents of the accumulator 0, 1, 2, 3, or 4 places depending upon the contents of the address portion of the instruction

8 BRANCH ON ZERO test the contents of the accumulator and branch to the indicated address if the contents of the accumulator is equal to zero

9 PRINT the contents of the accumulator (the address portion of a print statement is used as a flag on the output

The instructions '6000' and '7000' may be used as no-operation instructions.

The following is a higher-level language definition of the BRAIN-2 machine used in this example.

```
      LET L2 = M(L1)                    (1)
      LET L1 = L1 + 1                   (2)
      PUT (181,188,L2)                  (3)
      GET (185,185,L3)                  (4)
      GET (186,188,L4)                  (5)
      IF (L3) 7,7,9                     (6)
      STOP                              (7)
      GO TO 1                           (8)
      LET L11 = L3 + 900                (9)

1     READ (101,104,L6)                 (1)
      RETURN                            (2)

2     LET L6 = M(L4)                    (1)
      RETURN                            (2)

3     LET M(L4) = L6                    (1)
      RETURN                            (2)

4     LET L7 = M(L4)                    (1)
      LET L6 = L6 + L7                  (2)
      LET L9 = L6 - 10000               (3)
      IF (L9) 5,6,6                     (4)
      RETURN                            (5)
```

```
      LET L6 = L6 - 10000                    (6)
      LET L101 = 214                         (7)
      LET L102 = 229                         (8)
      LET L103 = 198                         (9)
      LET L104 = 211                        (10)
      LET L9 = 105                          (11)
      LET L(L9) = 64                        (12)
      LET L9 = L9 + 1                       (13)
      LET L10 = L9 - 115                    (14)
      IF (L10) 12,12,16                     (15)
      WRITE                                 (16)
      STOP                                  (17)
      LET L101 = 64                         (18)
      LET L102 = 64                         (19)
      LET L103 = 64                         (20)
      LET L104 = 64                         (21)
      WRITE                                 (22)
      RETURN                                (23)

5     LET L7 = M(L4)                         (1)
      LET L6 = L6 - L7                       (2)
      IF (L6) 5,4,4                          (3)
      RETURN                                 (4)
      LET L6 = 0                             (5)
      LET L101 = 228                         (6)
      LET L102 = 213                         (7)
      LET L103 = 198                         (8)
      LET L104 = 211                         (9)
      LET L9 = 105                          (10)
      LET L(L9) = 64                        (11)
      LET L9 = L9 + 1                       (12)
      LET L10 = L9 - 115                    (13)
      IF (L10)11,11,15                      (14)
      WRITE                                 (15)
      STOP                                  (16)
```

```
      LET L101 = 64                    (17)
      LET L102 = 64                    (18)
      LET L103 = 64                    (19)
      LET L104 = 64                    (20)
      WRITE                            (21)
      RETURN                           (22)

6     LET L7 = L4                      (1)
      IF (L7) 3,3,4                    (2)
      RETURN                           (3)
      PUT (191,198,L6)                 (4)
      LET L199 = L195                  (5)
      GET (196,199,L6)                 (6)
      LET L7 = L7 - 1                  (7)
      GO TO 2                          (8)

7     LET L7 = L4                      (1)
      IF (L7) 3,3,4                    (2)
      RETURN                           (3)
      LET L6 = L6/10                   (4)
      LET L7 = L7 - 1                  (5)
      GO TO 2                          (6)

8     IF (L6) 3,2,3                    (1)
      L1 = L4                          (2)
      RETURN                           (3)

9     PUT (108,115,L6)                 (1)
      PUT (101,108,L4)                 (2)
      LET L109 = 122                   (3)
      WRITE                            (4)
      RETURN                           (5)
      END
```

Following is a sample machine language program for the BRAIN-2 machine defined in this example:

```
M-000    1000
M-001    3008
M-002    1000
M-003    4008
M-004    9000
M-005    2007
M-006    8000
M-007    0000
```

The following FORTRAN-like program is roughly equivalent to the above machine language program:

```
 5  READ (10, END=20) A
10  FORMAT (I5)
    READ (10) B
    SUM = A + B
    Z = 0
    WRITE (15) Z, SUM
15  FORMAT (I8,":",I7)
    SUM = Z
    IF (SUM) 20,5,20
20  STOP
    END
```

MICRO-MEMORY CONTENTS

ADDR										
201	CCC10C18	CC180024	00250002	00010024	08930025	00250001	08980022	0897CC23	00020021	C8980022
211	C896CC23	CC21CCC3	C8950022	08950023	00210004	08940022	C8940023	0021CC05	C8930022	C8970023
221	CC210CC6	CCC6C024	CC050017	00160025	00250006	00030024	C8920025	CC25CC11	C0200CC0	C8900024
231	CC25CC4C	CC4CC024	08890025	00250041	00400022	00410023	C0060019	0021CC18	C0400024	C8880025
241	CC25CC4C	CCC6C024	08990025	00250006	00040017	00160024	08990027	C004C017	00270016	C0040017
251	CC160C24	CC15C024	00110025	00250011	08870011	08870011	C8860C11	0885CC11	08930022	C8920023
261	CCC60C19	0C180021	00000020	08850011	00060019	00180024	00040017	0025CC16	C8950011	00040017
271	CC160C24	0CC6C019	00250018	08850011	00040017	00160024	0006C01C	C018CC25	00040017	C0250016
281	C8850C11	CCC4C017	00160030	00060019	00180031	00040017	00310016	08850C11	00040017	00160026
291	CCC60C19	CC180027	00270C39	00390024	00150024	00110025	00250011	C876CC11	08750011	08740011
301	C873CC24	CC25C0C7	C8720011	08710024	00250007	08720011	08700024	CC25CCC7	C8950011	CC070026
311	C873CC27	CC27C038	0C380024	00150024	00110025	00250011	C8680011	0867C011	C8680011	0C060024
321	CC250CC1	C885C011	00000000	00000000	00000000	00000000	00000000	0CCCCCCC	00000000	C0000000

MICRO-MEMORY CONTENTS

ADDR										
801	CCCCCCOC	CCCCCOCO	00000000	00000000	00000000	00000000	00000000	00C0CCCC	00000000	0C000000
811	CCCCCCCC	CCCCCCCC	00000000	00000000	00000000	00000000	00000000	00C0CCCC	C0000000	CC000000
821	CCCCCCCC	COCCCOCO	00000000	00000000	00000000	00000000	C0000000	00C0CCCC	C0000000	CC000000
831	CCCOCOOC	CCCCCOCO	00000000	00000000	00000000	00000000	00000000	C0CCCCCC	00000000	CC000000
841	CCCCCCCC	CCCCCOCO	00C00000	00000000	00000000	00000000	00000000	C0CCCCCC	C0000000	0C000000
851	CCCCOCCC	OCCCCOCO	00000000	00000000	00000000	00000000	0C000000	C0C0CCCC	C0000000	CC000000
861	CCCCCOCC	CCCCCOCO	00000000	00000000	00000000	00000323	00000320	00CCC322	00000310	CC000013
871	CCCC0012	CCCCC3C9	00000011	00000307	00000304	00000301	00000289	00CCC282	C0000275	00000270
881	CCCOC265	CCCCC130	00000125	00000259	00000201	00000232	00000258	00C0C0C5	00000004	00000101
891	C0CC0229	C0CCC9C0	00000185	00000184	00000183	00000182	00000188	00C00181	00000001	99999999
901	C8910C11	C8840011	08810011	03800011	08790011	99999999	99999999	C878CC11	99999999	C8770011
911	C8659C11	99999999	99999999	08660011	99999999	99999999	99999999	99999999	99999999	99999999
921	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
931	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
941	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
951	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
961	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
971	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
981	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999
991	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	99999999	

MICRO-MEMORY CONTENTS

ADDR										
201	CCC30024	CC15C024	C0110025	00250011	08990011	08990011	08980011	C8970C24	C0250003	C0010019
211	CC18C024	CC250CC2	C0010024	08970025	00250001	08960C22	C8950023	0CC2CC21	C8960022	C8950023
221	CC21CC04	C8S4C022	08930C23	00210005	08920011	08910024	00250003	C8SCCC22	C8900023	00210004
231	C8S90022	C8S5CC23	00210005	00040024	00150024	00110025	C0250011	C8880C11	C8880011	08870011
241	CCC4C026	C886C027	00270004	00040024	00150024	00110025	00250011	C885C011	08840011	08830011
251	CCC7CC24	C8970C25	00250007	00050024	00070017	00250016	C8880011	00C7CC24	C8970025	0C250007
261	CCC5CC19	CC180024	0C070017	00250016	08880011	08820024	0C050025	0025CC11	C0070017	CC160024
271	CC25CCC9	C8910024	00070017	00250016	00070026	08970027	CC270007	0020CC0C	C8800024	C0250051
281	C8790024	CC25C052	C0090024	00150024	00110025	00250011	C8780011	0878CC11	08770011	0C070017
291	CC16CC24	CC25C010	C8910024	00070017	00250016	00070026	08970027	0C27CCC7	00510022	C0520023
301	CC1CCC19	CC21CC18	00510024	08760025	00250051	00520024	C8760025	CC25CC52	C0090026	C8970027
311	CC27CC09	C875C011	08880011	00070017	00160024	00250009	C8910024	00C7CC17	00250016	C0070026
321	C8970C27	0C27C007	08800024	00250051	08790024	00250052	C0090024	C015CC24	C0110025	C0250011
331	C8730011	C8730011	08720011	00070017	00160024	00250010	08910024	0CC7CC17	00250016	C0070026
341	C8970C27	CC27C0C7	00510022	00520023	00100019	00180021	00510024	C8760C25	00250051	C0520024
351	C876CC25	CC250052	C0090026	08970027	00270009	08710011	00000020	C8880C11	00070026	C8970027
361	CC27CCC6	CCC6C017	00160024	00250009	00070017	00160024	00090019	C025CC18	C8910024	00070017
371	CC250C16	C8910024	00060017	00250016	00070026	08860027	00270007	C888CC11	0C070024	00250006
381	CCC7CC24	C8970025	00250007	00060017	00160024	00070017	00250016	C888CC11	00070026	C8970027
391	CC27CCC6	CCC7C017	00160024	00250009	00060017	00160024	C007C017	0C25CC16	C0090024	C0060017
401	CC250C16	C888C011	08910024	00070017	00250016	00070026	08970027	C027CCC7	C8880011	00070026
411	C8970C27	CC27C0C6	C0060017	00160026	00070017	00160027	0C060017	0027CC16	08910024	00070017
421	CC250C16	CCC7C026	08970027	00270007	08880011	00070026	08970027	0027C006	C0060017	C0160028
431	CCC7CC17	CC16C029	C0060017	00290016	08910024	00070017	C0250016	00C7CC26	C8970027	00270007
441	C8880C11	C891C024	C0250003	00070017	00160024	00250001	C8910024	C0C7CC17	CC250015	C0070026
451	C8970C27	CC27CCC7	C8880011	00070026	08970027	00270006	CC060017	0016CC24	00150024	00110025
461	0C250C11	C862C011	08610011	08620011	08910024	00250003	C0070017	0016CC24	00250001	C8910024
471	CCC70C17	0C250016	00070026	08970027	00270007	08880011	00000000	C0000CCC	00000000	00000000

MICRO-MEMORY CONTENTS

ADDR										
801	0CCCCCCC	CCCCCCCC	00000000	00000000	00000000	00000000	00000000	00CCCCCC	00000000	00000000
811	CCCCCCCC	CCCCCCCC	C0000000	00000000	00000000	00000000	0CCCC000	0CCCCCCC	CCCC0000	CC000000
821	CCCCCCCC	CCCCCCCC	00000000	00000000	00000000	00000000	CC000000	CCCCCCCC	0C000000	00000000
831	CCCC000C	CCCCCCCC	00000000	00000000	00000000	00000000	00000000	00CCCCCC	00000000	C0000000
841	CCCCCCCC	CCCCCCCC	00000000	00000000	00000000	00000000	00000000	CCCCCCCC	C0000000	00000000
851	CCCCCC00	CCCCCCCC	00000000	00000000	00000000	00000000	00000000	CCCCCCCC	C00C0000	00000477
861	CCCCC46E	CCCCC470	00000454	00000442	00000426	00000410	00000403	00CCC3E9	C0000379	C0000359
871	CCCCC327	CCCC0334	00000357	00000314	00000293	00000010	00000290	00CC0213	C0000110	00000101
881	C0CC0269	CCCCC9C0	00000266	00000258	00000251	00000002	00000241	00CC02C1	000C0186	C0000185
891	CCCCCC00	CCC00234	00000184	00000182	00000188	00000181	C0000001	0000C226	C0000208	99999999
901	C8F10C11	C874C011	08700011	99999999	99999999	08690011	C8680011	08670C11	99999999	99999999
911	C864CC11	C863C011	99999999	99993999	99995939	99999999	99999999	99555559	99999999	59999999
921	99559959	C866C011	08650011	99999999	99599999	99999999	99999999	99555559	99999999	59999999
931	99559953	99559959	99999999	99993999	99999999	99999999	99599999	99555559	99999999	99999999
941	99559959	99559959	99995999	99999999	99599999	99999999	99999999	99555559	99999999	59999999
951	99559999	99559959	99999999	99999999	99999999	99999999	99599999	99555559	99999999	59999999
961	99559959	99559959	99999999	99999999	99999999	99999999	99999999	99555559	99999999	99999999
971	99559999	99559959	99999999	99993999	99999999	99999999	99999999	99555559	99993959	99999999
981	99559959	99559959	99999999	99993999	99999999	99999999	99999999	99559559	99999959	99999999
991	99559959	99559959	99999999	99999999	99999999	99999999	99999999	99559559	08600011	

DISPLAY PRINTOUT

L-CC1 0CC000C2	R-CCC CCCCC0C0	L-101 00000064	M-000 03200013
L-002 0120002C	R-CC1 CCCC00C0	L-102 00000064	M-001 01200020
L-003 0CC00CC1	R-CC2 CCCCC000	L-103 00000241	M-002 03300011
L-004 CCC00002	R-CC3 CC0CC0C0	L-104 00000240	M-003 05130020
L-005 CCCC000C	R-0C4 CC0CC000	L-105 00000240	M-004 05300012
L-006 CC000024	L-231 CC25C040	L-106 00000064	M-005 10300013
L-007 CCCC00CC	L-181 CCC0C064	L-107 00000064	M-006 11000003
L-008 C0CCC0CC	L-182 CCC0C241	L-108 00000241	M-007 08100013
L-009 0CC00000	L-183 C000C242	L-109 00000245	M-008 04100024
L-01C 0CCC0CCC	L-184 CC0CC240	L-110 00000240	M-009 02000024
L-011 00C00201	L-185 CCCCC240	L-111 00000064	M-010 14000000
L-012 C8850C11	L-186 CC0CC240	L-112 00000064	M-011 00000000
L-013 CCC00895	L-187 CCCCC242	L-113 00000242	M-012 00000001
L-014 0C000C11	L-188 CCCCC240	L-114 00000240	M-013 00000004
L-015 CCC00C03		L-115 00000240	M-014 00000000
L-016 CCCCCCCC		L-116 00000064	M-015 00000000
L-017 0CC00C02		L-117 00000064	M-016 00000000
L-018 00000250		L-118 00000242	M-017 00000000
L-019 CCC00023		L-119 00000245	M-018 00000000
L-02C CCC00CCC		L-120 00000240	M-019 00000000
L-021 0000025C		L-121 00000064	M-020 00000100
L-022 CC000116		L-122 00000064	M-021 00000150
L-023 C000012C		L-123 00000064	M-022 00000200
L-024 CCC00C02		L-124 00000064	M-023 00000250
L-025 0CC00256		L-125 00000064	M-024 00000000

INPUT * DISPLAY L2C1-3CC,1

OUTPUT

KEYBOARD L24C-25C,C

DISPLAY PRINTOUT

L-001 0C0C0C1C	R-CCC CCCCC0C0	L-101 00000064	M-000 03200013
L-002 02000024	R-CC1 CC000175	L-102 00000064	M-001 01200020
L-0C3 0CC00CC2	R-CC2 CCCCC0C0	L-103 00000241	M-002 03300011
L-004 CCC0CCCC	R-CC3 CC00000C4	L-104 00000240	M-003 05130020
L-005 CCCC0CCC	R-0C4 CCCCCCC0	L-105 00000240	M-004 05300012
L-006 C0000C24	L-231 CC25C040	L-106 00000064	M-005 10300013
L-007 CCC00012	L-181 CC00C064	L-107 00000064	M-006 11000003
L-0C8 0CC00CCC	L-182 CCC0C242	L-108 00000241	M-007 08100013
L-009 0CC000CC	L-183 C0CC0240	L-109 00000245	M-008 04100024
L-01C 0CCC0CCC	L-184 0C0C0240	L-110 00000240	M-009 02000024
L-011 CCC002C1	L-185 CCCCC240	L-111 00000064	M-010 14000000
L-012 08850011	L-186 CC0CC240	L-112 00000064	M-011 00000000
L-013 0C000885	L-187 CC000242	L-113 00000242	M-012 00000001
L-014 0CC00011	L-188 CCC0C244	L-114 00000240	M-013 00000004
L-015 0CC000C3		L-115 00000240	M-014 00000000
L-016 0CC00000		L-116 00000064	M-015 00000000
L-017 CCCCCCCC		L-117 00000064	M-016 00000000
L-018 CC000175		L-118 00000242	M-017 00000000
L-C19 CC000024		L-119 00000245	M-018 00000000
L-02C 0CCCC00C		L-120 00000240	M-019 00000000
L-021 C0000175		L-121 00000064	M-020 00000100
L-022 0C000125		L-122 00000064	M-021 00000150
L-023 0C00013C		L-123 00000064	M-022 00000200
L-024 0C000002		L-124 00000064	M-023 00000250
L-025 00C00902		L-125 00000064	M-024 00000175

INPUT $ DISPLAY L2C1-3CC,1

OUTPUT 1CC 15C 2CC 25C 175

KEYBOARD L24C-25C,C

```
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION ST(100),RESWD(8),CD(2,4),DATA(80),CHAR(10)
      DIMENSION INP(8), OUP(8), KEYB(8),NULL(1)
      LOGICAL*1 DSPLY,BLANK1,OUTPUT,INP,OUP,KEYB,LOUT(100,80)
      LOGICAL*1 LST(72), LTEMP,DUM
      LOGICAL*1 LSTR(80),LSTC(80),LLL(4)
C
      COMMON/EE/ DSPLY(72,28), LCW, UP
      COMMON /ER/ STR(80),KCL,NBR,OPR
      COMMON/EE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
      COMMON /EL/ VRB,VSUB,LC,LCP
      COMMON /EI/ LINST,OUTPUT(8),DUM(24)
      COMMON /EC/ RZ,R(99),MZ,M(999)
      COMMON /EH/ DG
      COMMON/JP/ NULL,IGD3
      COMMON/EB/ LPIC(999), RPICZ, RPIC(99), MPICZ, MPIC(999)
      COMMON/JQ/ CTZ,OT(999)
C
      EQUIVALENCE (TEMP,LTEMP)
      EQUIVALENCE (LLLTMP,LLL)
C
      DATA RESWD/'R','G','W','P','L','I','S','F'/,A/'A'/,
     2 E/'E'/,CD/24,25,26,27,28,29,30,31/
C
      DATA CHAR/'N','D','3','4','5','6','7','8','9','1'/
C
      DATA BLANK1/Z40/,LL/'L'/,RR/'R'/,MM/'M'/,INP/'I','N','P','U','T',
     1' ',' ',' '/,OUP/'O','U','T','P','U','T',' ',' '/,KEYB/'K','E',
     2'Y','B','C','A','R','D'/
C
      DATA CC/'C'/,DD/'D'/,PP/'P'/,SS/'S'/,OO/'O'/
C
      DATA TEMP/'    '/, BLNK/' '/
      DATA LLLTMP/0/, DOLLAR/'$'/
C
C     INITIALIZATION
C
      Z=0
      DO 498C I = 1,999
 498C L(I) = C
      DO 4985 I = 101,180
 4985 L(I) = 64
      DO 499C I=Z,99
 499C R(I)=C
      DO 4995 I=Z,999
 4995 M(I)=C
      DO 4998 I=1,100
```

```
      DO 4997 J = 1,80
      LOUT(I,J)=BLANK1
 4997 CONTINUE
 4998 CONTINUE
C
CC
CC    INITIALIZE GSP AND GRAPHIC DATA SETS
CC
C
      IC=3
      NULL(1)=5
      CALL INGSP(IGSP,NULL)
      CALL INDEV(IGSP,40,I2250)
      CALL INGDS(I2250,IGD3)
      CALL INGDS(I2250,IGD1)
      CALL SDATM(IGD1,IC)
      CALL SDATM(IGD3,IC)
      CALL SCHAM(IGD3,3)
      CALL SALRM(I2250)
      CALL CRATL(I2250,IATN)
      CALL ENATN(IATN,1,-9,20,-30,32,34)
C
CC   SET UP BOXES ON THE SCREEN
C
      CALL PSGMT(IGD1,315,500,315,3750,NULL,NULL,NULL,4,NULL,NULL,NULL,
     1NULL,1060,0,1060,0)
      CALL PSGMT(IGD1,865,500,865,3750,NULL,NULL,NULL,4,NULL,NULL,NULL,
     1NULL,1060,0,1060,0)
      CALL PSGMT(IGD1,315,500,865,500,NULL,NULL,NULL,26,NULL,NULL,NULL,
     1NULL,0,130,0,130)
      CALL PSGMT(IGD1,1375,500,1925,500,NULL,NULL,NULL,26,NULL,NULL,
     1NULL,NULL,0,130,0,130)
      CALL PSGMT(IGD1,2435,500,2985,500,NULL,NULL,NULL,26,NULL,NULL,
     1NULL,NULL,0,130,0,130)
      CALL PSGMT(IGD1,3495,500,4045,500,NULL,NULL,NULL,26,NULL,NULL,
     1NULL,NULL,0,130,0,130)
      CALL PSGMT(IGD1,490,20,4090,20,NULL,NULL,NULL,4,NULL,NULL,NULL,
     1NULL,0,140,0,140)
      CALL PSGMT(IGD1,490,20,490,440,NULL,NULL,NULL,2,NULL,NULL,NULL,
     1NULL,3600,0,3600,0)
C
CC    INITIALIZE DISPLAY BOX ELEMENTS
C
      DO 1000 I = 2,99
 1000 RPIC(I) = 0
      DO 1001 I = 2,999
 1001 MPIC(I) = 0
      DO 1002 I=1,999
```

```
 1002 LPIC(I) = C
C
      DO 1010 I=1,28
      DO 1005 J=1,72
      DSPLY(J,I) = BLANK1
 1005 CONTINUE
 1010 CONTINUE
C
C
C    INITIALIZE THE 28 DISPLAY ELEMENTS
C
 1055 DO 1072 I=1,25
      CORV = I
      XX = 40
      YY = 3685-(130*(I-1))
      CALL PTEXT(IGD3,DSPLY(1,I),72,CORV,NULL,1,XX,YY)
 1072 CONTINUE
C
      DO 1056 I=1,8
      DSPLY(I,26) = INP(I)
      DSPLY(I,27) = OUP(I)
      DSPLY(I,28) = KEYB(I)
 1056 CONTINUE
      CALL PTEXT(IGD3,DSPLY(1,26),72,26,NULL,1,40,370)
      CALL PTEXT(IGD3,DSPLY(1,27),72,27,NULL,1,40,230)
      CALL PTEXT(IGD3,DSPLY(1,28),72,28,NULL,1,40,90)
C
C    EXECUTE THE PROGRAMMER INITIALIZED DISPLAY
C
   71 CALL EXEC(IGD3)
      CALL EXEC(IGD1)
C
      CALL ICLRS(IGD3,28,NULL,11)
      GOTO 8200
C
C
C    READ AND DISPLAY NEXT CARD
C
C
 8200 READ(5,8210,END=8285) LSTR
 8210 FORMAT(80A1)
      DO 8282 I = 11,72
      DSPLY(I,26) = LSTR(I-10)
 8282 CONTINUE
      CALL PTEXT(IGD3,DSPLY(1,26),72,26,NULL,3,40,370)
      GOTO 4094
C
 8285 CALL PRDIAG(4,&4094)
```

```
C
CCC
CCC    PROCESS CONTROL CARD
CCC
C
 1500 DO 1501 I=1,80
      LTEMP=LSTR(I)
      STR(I)=TEMP
 1501 CONTINUE
      IF (STR(1).NE.DOLLAR) CALL PRDIAG(5,&4094)
      KOL=2
      CALL BLANK
      IF (STR(KOL).EQ.CC) GO TO 1800
      IF (STR(KOL).EQ.RP) GO TO 4950
      IF (STR(KOL).EQ.DD) GO TO 1020
      IF (STR(KOL).EQ.PP) GO TO 1520
      CALL PRDIAG(6,&4094)
C
CCC
CCC   BEGIN COMPILER
CCC
C
 1800 DO 1810 I=1,899
      IF (I.LT.101.OR.I.GT.180) L(I)=0
      IF (I.LT.181.AND.I.GT.100) L(I)=64
 1810 CONTINUE
C
      WRITE(6,1811)
 1811 FORMAT(1H1)
C
      DO 1820 I=900,999
      L(I)=99999999
 1820 CONTINUE
      LC=900
      LA=200
      MXLOP=899
      S=0
      RE=0
      WR=0
      IFGO=0
      SSW=0
      OSW=0
      DG=0
C
 1825 CALL NTBLNK
      CALL BLANK
      IF (KOL.EQ.80) GO TO 2000
```

```
      IF (STR(KCL).EQ.SS) SSW=1
      IF (STR(KCL).EQ.CO) OSW=1
      GO TO 1825
C
 2000 IF (DG.EQ.1) CALL PRDIAG(3,64094)
      READ(5,2001) STR
 2001 FORMAT (80A1)
      IF(SSW.EQ.1) WRITE(6,2002) STR
 2002 FORMAT(1HO,45X,80A1)
C
      S=S+1
      ST(S)=LA+1
      DO 2010 I=1,7
 2010 LIN(I)=0
C
      KOL=1
 2020 CALL BLANK
      DO 2030 I=1,8
      IF (STR(KOL).EQ.RESWD(I)) GO TO (2041,2042,603,604,
     1 605,606,609,8200),I
 2030 CONTINUE
C
C
 2033 IF (IFGO.EQ.0) GO TO 2036
      DO 2035 I=LC,MXLCP
      IF (L(I).LT.0) L(I)=ST(IABS(L(I)))
 2035 CONTINUE
C
 2036 MXLCP=LC
      CALL GETNUM
      NBER=NBR+900
      NBR=LA+1
      CALL GETCON
      L(NBER)=LCP*10000+11
      CALL UPDATE(1,NBER)
      S=1
      ST(S)=LA+1
      IFGO=0
      GO TO 2020
C
 2041 KOL=KOL+2
      IF (STR(KOL).EQ.A) GO TO 601
      GO TO 608
 2042 KOL=KOL+1
      IF (STR(KOL).EQ.E) GO TO 602
      GO TO 607
C
C     READ STATEMENT
```

```
C
  601 CALL PRCVAR(&2050)
      CALL MKLIN(1,2,1,22)
      CALL MKLIN(1,4,2,23)
      CALL MKLIN(2,6,3,21)
      IF (RE.EQ.1) GO TO 2060
 2050 LIN(1)=200000
 2060 CALL MKCRD(7)
      RE=0
      GO TO 2000
C
C     GET STATEMENT
C
  602 RE=1
      GO TO 601
C
C     WRITE STATEMENT
C
  603 CALL PRCVAR(&2070)
      CALL MKLIN(1,1,1,22)
      CALL MKLIN(1,3,2,23)
      CALL MKLIN(1,5,3,21)
      IF (WR.EQ.1) GO TO 2080
 2070 LIN(7)=20
 2080 CALL MKCRD(7)
      WR=0
      GO TO 2000
C
C     PUT STATEMENT
C
  604 WR=1
      GO TO 603
C
C     LET STATEMENT
C
  605 CALL NTBLNK
      CALL LETT
      IF (OPR.LT.5) GO TO 2110
      OPR=1
      GO TO 2120
C
 2110 CALL MKLIN(1,3,3,CD(2,OPR))
 2120 CALL MKLIN(1,1,2,CD(1,OPR))
      CALL MKLIN(2,5,1,CD(2,OPR))
C
      CALL MKCRD(6)
      GO TO 2000
C
```

```
C       IF STATEMENT
C
  6C6 KCL=KCL+2
      CALL BLANK
      KCL=KCL+1
      CALL GTCPND
      VAR(1)=VRB
      VS(1)=VSUB
      CALL MKLIN(1,1,1,24)
C
      DO 2140 I=1,3
      CALL GETNUM
      NBR=-NBR
      CALL GETCON
      LIN(I+5)=LCP*10000+11
      CALL BLANK
      KCL=KCL+1
 2140 CONTINUE
C
      IFGO=1
      LIN(3)=150024
      LIN(4)=110025
      LIN(5)=250011
      CALL MKCRD(8)
      GO TO 2000
C
C       GO TO STATEMENT
C
  6C7 CALL GETNUM
      NBR=-NBR
      CALL GETCON
      LIN(1)=LCP*10000+11
      CALL MKCRD(1)
      IFGO=1
      GO TO 2000
C
C       RETURN STATEMENT
C
  6C8 NBR=2C1
      CALL GETCON
      LIN(1)=LCP*10000+11
      CALL MKCRD(1)
      GO TO 2000
C
C       STOP STATEMENT
C
  6C9 LA=LA+1
      IF (LA.EQ.LC) CALL PRDIAG(3,&4094)
```

```
      L(LA)=C
      CALL UPDATE(1,LA)
      IF (DSW.EQ.1) WRITE(6,2143) LA,L(LA)
 2143 FORMAT(50X,I3,2X,I8)
      GOTO 2000
C
CC
CCC
CCCC
C    END OF COMPILER
CCCCC
CCCCCC
CCCCCCC
CCCC
CCC
C
 4950 CALL GETNLU(N)
      GO TO (4951,4952,4953),N
C
 4951 READ(5,4955) (L(I),I=LOW,UP)
 4955 FORMAT(8I10)
      DO 4961 I=LOW,UP
      CALL UPDATE(1,I)
 4961 CONTINUE
      GO TO 8200
C
 4952 READ(5,4955) (R(I),I=LOW,UP)
      DO 4962 I=LOW,UP
      CALL UPDATE(2,I)
 4962 CONTINUE
      GO TO 8200
C
 4953 READ(5,4955) (M(I),I=LOW,UP)
      DO 4963 I=LOW,UP
      CALL UPDATE(3,I)
 4963 CONTINUE
      GO TO 8200
C
CC
C    READ IN DISPLAY BOX DEFINITIONS
C
 1020 CALL NTBLNK
      CALL BLANK
      IF(STR(KOL).EQ.LL) GOTO 1030
      IF (STR(KOL).EQ.RR) GO TO 1040
      IF (STR(KOL).EQ.MM) GO TO 1050
```

```
C
 1030 CALL GETPIC(LPIC,STR(KCL))
      N = 1
      GOTO 1051
C
 1040 CALL GETPIC(RPIC,STR(KCL))
      N = 2
      GOTO 1051
C
 1050 CALL GETPIC(MPIC,STR(KCL))
      N = 3
C
 1051 DO 1053 I = LOW,UP
      CALL UPDATE(N,I)
 1053 CONTINUE
      GOTO 8200
C
C
C    PRINT L, R, OR M
C
C
 1520 CALL GETNLU(N)
      GO TO (1530,1540,1550),N
C
 1530 DO 1535 I=LOW,UP
      OT(I)=L(I)
 1535 CONTINUE
      WRITE(6,1536)
 1536 FORMAT(1H1////65X,'MICRO-MEMORY'/15X,'ADDR',
     1 59X,'CONTENTS'/)
      GO TO 5003
C
 1540 DO 1545 I=LOW,UP
      OT(I)=R(I)
 1545 CONTINUE
      WRITE(6,1546)
 1546 FORMAT(1H1////65X,'GENERAL PURPOSE REGISTERS'/15X,'REG NO',
     1 63X,'CONTENTS'/)
      GO TO 5003
C
 1550 DO 1555 I=LOW,UP
      OT(I)=M(I)
 1555 CONTINUE
      WRITE(6,1556)
 1556 FORMAT(1H1//65X,'MAIN MEMORY'/15X,'ADDR',57X,'CONTENTS'/)
C
 5003 ROWNO=(UP-LOW)/10+1
      DO 5340 J=1,ROWNO
```

```
      ADDR=LOW
      IA=0
C
      DO 5330 I=1,10
      CALL CONVRT(CT(LOW),8)
      LOW=LOW+1
      DO 5320 LG=1,8
      IA=IA+1
      LOUT(J,IA) = OUTPUT(LG)
 5320 CONTINUE
      IF ((LOW-1).EQ.UP) GO TO 5333
 5330 CONTINUE
C
 5333 CALL CONVRT(ADDR,3)
 5335 WRITE(6,5336) (OUTPUT(I),I=1,3),(LOUT(J,K),K=1,80)
 5336 FORMAT(15X,3A1,2X,10(2X,8A1)/)
C
      DO 5338 BL=1,80
      LOUT(J,BL)=BLANK1
 5338 CONTINUE
 5340 CONTINUE
      GO TO 8200
C
C
C    INTERPRETER
C
C
 4090 L(12) = L(L(11))
      CALL UPDATE(1,12)
      IF(L(12).EQ.0) GOTO 4094
      L(11) = L(11)+1
      CALL UPDATE(1,11)
C
      CALL GETCRD(L(12),L(13),L(14))
      CALL UPDATE(1,13)
      CALL UPDATE(1,14)
C
      IF (L(13).GT.999) CALL PRDIAG(2,&4094)
      IF (L(13).GT.32) GO TO 801
      IF (L(13).EQ.0) GO TO 5120
C
C     FETCH CONTENTS OF MICROMEMORY LOCATION AS SPECIFIED BY
C        L13 AND LOAD INTO THE TRANSFER REGISTER
C
      L13=L(13)
      GO TO (801,801,801,801,801,801,801,801,801,801,801,801,801,801,
     1 801,816,801,818,801,820,821,801,801,801,801,801,801,801,801,801,
     2 801,832),L13
```

```
C
  801 TREG=L(L(13))
      GO TO 5120
C
  816 L(16)=R(L(17))
      CALL UPDATE(1,16)
      GO TO 801
C
  818 L(18)=M(L(19))
      CALL UPDATE(1,18)
      GO TO 801
C
  820 LTEMP=LSTR(1)
      IF (TEMP.EQ.DOLLAR) CALL PRDIAG(1,&4094)
      DO 5115 I = 101,180
      LLL(4) = LSTR(I-100)
      L(I) = LLLTMP
 5115 CALL UPDATE(1,I)
      READ(5,8021,END=8285) LSTR
 8021 FORMAT(80A1)
      DO 8288 I = 11,72
      DSPLY(I,26) = LSTR(I-10)
      CALL PTEXT(IGD3,DSPLY(1,26),72,26,NULL,3,40,370)
 8288 CONTINUE
      GO TO 5120
C
  821 IN=0
      SIGN = 1
      IL=L(22)
      IU=L(23)
      DO 5118 I=IL,IU
      BUF = 0
      IF(L(I).EQ.96) SIGN = -1
      IF(L(I).GE.240) BUF = L(I)-240
 5105 IN = IN*10+BUF
 5118 CONTINUE
      L(L(13)) = SIGN*IN
      CALL UPDATE(1,L(13))
      GO TO 801
C
  832 L(32)=L(L(33))
      CALL UPDATE(1,32)
      GO TO 801
C
 5120 IF (L(14).GT.32) GO TO 901
      IF (L(14).EQ.0) GO TO 5100
C
C     STORE CONTENTS OF TRANSFER REGISTER IN THE LOCATION
```

```
C         SPECIFIED BY L14
C
      L14=L(14)
      GO TO (901,901,901,901,901,901,901,901,901,901,901,901,901,901,
     1 901,916,901,918,901,920,921,901,901,924,925,924,927,924,929,924,
     2 931,932),L14
C
  901 L(L(14))=TREG
      CALL UPDATE(1,L(14))
      GO TO 5100
C
  916 R(L(17))=TREG
      CALL UPDATE(2,L(17))
      GO TO 901
C
  918 M(L(19))=TREG
      CALL UPDATE(3,L(19))
      GO TO 901
C
  920 DO 8255 I = 101,180
      LLLTMP = L(I)
 8255 LSTO(I-100) = LLL(4)
      DO 8266 I = 11,72
 8266 DSPLY(I,27) = LSTO(I-10)
      CALL PTEXT(IGD3,DSPLY(1,27),72,27,NULL,3,40,230)
      WRITE(6,8304) LSTO
 8304 FORMAT(1H1,5X,'OUTPUT = ',80A1)
      GOTO 5100
C
  921 L(L(14))=TREG
      CALL UPDATE(1,L(14))
      BUF = IABS(L(L(14)))
      SIGN = 64
      IF(L(L(14)).LT.0) SIGN = 96
      IL=L(22)
      IU=L(23)
C
      DO 5125 I=IL,IU
      OUT=BUF/10
      IF (BUF.EQ.0) L(IL+(IU-I))=SIGN
      IF (BUF.EQ.0) SIGN=64
      IF (BUF.NE.0) L(IL+(IU-I))=BUF-OUT*10+240
      CALL UPDATE(1,IL+(IU-I))
 5125 BUF=OUT
      GO TO 5100
C
  924 L(L(14))=TREG
      CALL UPDATE(1,L(14))
```

IF(L(IU).EQ.64) L(IU)=240

```
      L(L(14)+1)=TREG
      CALL UPDATE(1,L(14)+1)
      GO TO 5135
C
  925 L(25)=L(25)+TREG
      CALL UPDATE(1,25)
      GO TO 5135
C
  927 L(27)=L(27)-TREG
      CALL UPDATE(1,27)
      GO TO 5135
C
  929 L(29)=L(29)*TREG
      CALL UPDATE(1,29)
      GO TO 5135
C
  931 L(31)=L(31)/TREG
      CALL UPDATE(1,31)
C
 5135 IF (L(L(14)).LT.0) L(15)=1
      IF (L(L(14)).EQ.0) L(15)=2
      IF (L(L(14)).GT.0) L(15)=3
      CALL UPDATE(1,15)
      GO TO 5100
C
  932 L(L(33))=TREG
      CALL UPDATE(1,L(33))
      GO TO 901
C
C
C     END OF INTERPRETER
C
C
 5100 IF(MODE.EQ.1.AND.L(11).NE.201) GOTO 4222
      IF(MODE.EQ.2) GOTO 4222
C
 4094 CALL MLITS(IATN,4,1,2,3,5,7,9,22,25,27,30)
      GOTO 44
C
C    SETS SINGLE INSTRUCTION MODE
C
 4088 MODE = 1
      CALL MLITS(IATN,4,4)
      GOTO 4090
C
C    SETS SINGLE CYCLE MODE
C
 4087 MODE = 0
```

```
      GOTO 4090
C
C    SETS 'GO' MODE
C
 4089 MODE = 2
      CALL MLITS(IATN,4,4)
      GOTO 4090
C
C    CHECK FOR ACTIVATION OF 'STOP' DURING 'GO' OR 'SINGLE
C    INSTRUCTION MODES
C
 4222 CALL RQATN(IATN,AKY,1,NULL,4)
      IF(AKY.EQ.4) GOTO 4094
      GOTO 4090
C
C    INSERT DATA VIA KEYBOARD
C
 1113 DO 1114 I = 1,80
      STR(I) = BLNK
 1114 CONTINUE
      CALL GSPRD(IGD3,LST,-72,1,TERK,28)
      KTERK = IABS(TERK)
C
      DO 4300 I = 11,KTERK
      DSPLY(I,28)=LST(I)
      LTEMP = LST(I)
 4300 STR(I-10) = TEMP
C
      KOL = 1
      CALL BLANK
      NSTR = STR(KCL)
      CALL GETNUM
      NR = IABS(NBR)
C
      IF(NSTR.EQ.LL) NA = 1
      IF(NSTR.EQ.RR) NA = 2
      IF(NSTR.EQ.MM) NA = 3
C
      CALL GETNUM
      IF(NA.EQ.1) L(NR) = NBR
      IF(NA.EQ.2) R(NR) = NBR
      IF(NA.EQ.3) M(NR) = NBR
C
      CALL UPDATE(NA,NR)
      GOTO 4094
C
C    INSERT LABELS VIA KEYBOARD
C
```

```
 1180 DO 1115 I = 1,80
      STR(I) = BLNK
 1115 CONTINUE
      CALL GSPRD(IGD3,LST,-72,1,TORK,28)
      KTORK = IABS(TORK)
C
      DO 4330 I = 11,KTORK
      DSPLY(I,28) = LST(I)
      LTEMP = LST(I)
 4330 STR(I-10) = TEMP
C
      KOL = 1
      CALL BLANK
      IF(STR(KOL).EQ.LL) GOTO 4333
      IF(STR(KOL).EQ.RR) GOTO 4335
      IF(STR(KOL).EQ.MM) GOTO 4337
C
 4333 CALL GETPIC(LPIC,STR(KOL))
      N = 1
      GOTO 4340
C
 4335 CALL GETPIC(RPIC,STR(KOL))
      N = 2
      GOTO 4340
C
 4337 CALL GETPIC(MPIC,STR(KOL))
      N = 3
C
 4340 DO 4350 I=LOW,UP
      CALL UPDATE(N,I)
 4350 CONTINUE
      GOTO 4094
C
C    PRINT CONTENTS OF DSPLY ARRAY
C
 4100 WRITE(6,4101) ((DSPLY(I,J),I=1,72),J=1,28)
 4101 FORMAT(1H1,//T35,'DISPLAY PRINTOUT',////(1HO,T8,72A1))
      GOTO 4094
C
C
C    RESET
C
C
 4230 L(11) = 201
      CALL UPDATE(1,11)
      DO 4232 K = 1,10
      L(K) = 0
      CALL UPDATE(1,K)
```

```
 4232 CONTINUE
      GOTO 4C94
C
  14  CALL MLITS(IATN,4,30)
C
C    ALL FUNCTION KEY AND KEYBOARD INTERRUPTS ARE PROCESSED THROUGH
C     THE ROATN SUBROUTINE
C
  44  CALL ROATN(IATN,IKY,2,NULL,1,-9,20,-30,32,34)
C
      GOTO(150C,82C0,4089,99,4087,99,4088,99,4230,99,99,99,99,99,99,99,
     199,99,99,4094,99,4100,99,99,1180,99,1113,99,99,102,99,99,99,
     299),IKY
C
  99  GOTO 4C94
C
C
C    TERMINATE THE DISPLAY DEVICE AND THE PROGRAM
C
 102  CALL TMDEV(I2250)
3CCC  STOP
      END


      SUBROUTINE UPDATE(N,IS)
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 NAME(3),DSPLY,OUTPUT,DUM
      LOGICAL*1 MINUS
      COMMON/JP/ NULL,IGD3
      COMMON /FI/ LINST,OUTPUT(8),DUM(24)
      COMMON/FF/ DSPLY(72,28), LOW, UP
      COMMON/FB/ LPIC(999), RPICZ, RPIC(99), MPICZ, MPIC(999)
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
      COMMON/FC/ RZ, R(99), MZ, M(999)
      DATA NAME/'L','R','M'/,MINUS/'-'/
C
      GO TO (3601,3602,3603),N
C
 3601 Q=LPIC(IS)
      P=L(IS)
      GO TO 3605
C
 3602 Q=RPIC(IS)
      P=R(IS)
      GO TO 3605
C
 3603 Q=MPIC(IS)
      P=M(IS)
```

```
C
 3605 IF (Q.EC.0) RETURN
C
      COL=(Q-1)/25+1
      ROW=Q-(CCL-1)*25
      POS=19*(COL-1)+1
C
      DSPLY(PCS,ROW)=NAME(N)
      DSPLY(PCS+1,ROW)=MINUS
      J = IS
      CALL CCNVRT(J,3)
      DSPLY(PCS+2,ROW) = OUTPUT(1)
      DSPLY(PCS+3,ROW) = OUTPUT(2)
      DSPLY(PCS+4,ROW) = OUTPUT(3)
      CALL CCNVRT(P,8)
      PC = PCS+5
      DO 3610 I=1,8
      DSPLY(PC+I,RCW) = CUTPUT(I)
 361C CONTINUE
C
      YY=3685-(130*(ROW-1))
      CORV=RCW
      CALL PTEXT(IGD3,DSPLY(1,ROW),72,CORV,NULL,3,40,YY)
      RETURN
      END


      SUBROUTINE GETPIC(PIC,XSTR)
      IMPLICIT INTEGER (A-Z)
      DIMENSION PIC(2)
      LOGICAL*1 NAM(4), DSPLY, MINUS, OUTPUT,DUM
      LOGICAL*1 BLANK1
C
      COMMON/JP/ NULL,IGD3
      COMMON/FE/ DSPLY(72,28), LCW, UP
      COMMON/FR/ STR(80), KOL, NBR, OPR
      COMMON/FI/ LINST, CUTPUT(8), DUM(24)
      COMMON/FB/ LPIC(999), RPICZ, RPIC(99), MPICZ, MPIC(999)
C
      EQUIVALENCE (STRING,NAM)
      DATA MUS/'-'/, MINUS/Z60/
      DATA BLANK1/Z40/
C
      Z = 0
      STRING = XSTR
      CALL GETNUM
      LOW = IABS(NBR)
      IF(STR(KOL).EQ.MUS) CALL GETNUM
```

```
      UP = IABS(NBR)
      CALL GETNUM
      BXNO = NBR
C
      IF(BXNO.EQ.0) GOTO 3553
C
      DIFF = UP - LOW
      HIGH = BXNO + DIFF
      DO 9230 J = 1,999
      IF(LPIC(J).GE.BXNO.AND.LPIC(J).LE.HIGH) LPIC(J) = 0
 9230 CONTINUE
      DO 9250 J = 1,999
      IF(MPIC(J).GE.BXNO.AND.MPIC(J).LE.HIGH) MPIC(J) = 0
 9250 CONTINUE
      DO 9270 J = 1,99
      IF(RPIC(J).GE.BXNO.AND.RPIC(J).LE.HIGH) RPIC(J) = 0
 9270 CONTINUE
C
C
 9004 DO 3500 I = LOW,UP
      COL = (BXNO-1)/25+1
      ROW = BXNO-(COL-1)*25
      POS = 19*(COL-1)+1
C
      DSPLY(POS,ROW) = NAM(1)
      DSPLY(POS+1,ROW) = MINUS
      CALL CONVRT(I,3)
      DSPLY(POS+2,ROW) = OUTPUT(1)
      DSPLY(POS+3,ROW) = OUTPUT(2)
      DSPLY(POS+4,ROW) = OUTPUT(3)
C
      PIC(I) = BXNO
      BXNO = BXNO+1
 3500 CONTINUE
      RETURN
C
 3553 DO 3554 K = LOW,UP
      BNO = PIC(K)
      COL = (BNO-1)/25+1
      ROW = BNO-(COL-1)*25
      POS = 19*(COL-1)+1
      SOP = POS+13
C
      DO 9100 JK = POS,SOP
 9100 DSPLY(JK,ROW) = BLANK1
C
      YY = 3685-(130*(ROW-1))
      CALL PTEXT(IGD3,DSPLY(1,ROW),72,ROW,NULL,3,40,YY)
```

```
      PIC(K) = 0
 3554 CONTINUE
C
      RETURN
      END


      SUBROUTINE CONVRT(WORD,LEN)
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DIV(9),DIGIT(8),DUMMY(8)
      LOGICAL*1 OUTPUT, LDIG(32),DUM
      LOGICAL*1 MS/Z60/
      COMMON /FI/ LINST,OUTPUT(8),DUM(24)
      EQUIVALENCE (DIGIT,LDIG), (DUMMY,OUTPUT)
C
      DIV(1)=10**LEN
      LL=LEN+1
      WORRD = IABS(WORD)
C
      DO 3045 I=2,LL
 3045 DIV(I)=DIV(I-1)/10
C
      DO 3050 I=1,LEN
 3050 DIGIT(I) = ((WORRD-((WORRD/DIV(I))*DIV(I)))/DIV(I+1)+240
C
      DO 3060 I=1,LEN
 3060 OUTPUT(I)=LDIG(4*I)
      IF(WORD.LT.0) OUTPUT(1) = MS
C
      RETURN
      END


      SUBROUTINE GETNUM
      IMPLICIT INTEGER (A-Z)
      DIMENSION DIGIT(10)
      COMMON /FR/ STR(80),KCL,NBR,OPR
      DATA DIGIT/'0','1','2','3','4','5','6','7','8','9'/,
     2 MINUS/'-'/
C
      FL=0
      NBR=0
      SIGN=1
      KOLL = KCL
 3410 DO 3420 NO=1,10
      IF (STR(KCL).EQ.DIGIT(NO)) GO TO 3430
 3420 CONTINUE
```

```
C
      IF (FL.EQ.0) GO TO 3440
      NBR=NBR*SIGN
      RETURN
C
 3430 NBR=NBR*10+NC-1
      IF(FL.EQ.0.AND.KOL.GT.KOLL.AND.STR(KOL-1).EQ.MINUS) SIGN=-1
      FL=1
 3440 KOL=KOL+1
      GO TO 3410
      END
C


      SUBROUTINE BLANK
C
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      DATA BLNK/Z40404040/
C
 3450 IF (STR(KOL).NE.BLNK) RETURN
      IF (KOL.EQ.80) RETURN
      KOL=KOL+1
      GO TO 3450
      END


      SUBROUTINE NTBLNK
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      DATA BLNK/' '/
C
 3460 IF (STR(KOL).EQ.BLNK) RETURN
      KOL=KOL+1
      GO TO 3460
      END


      SUBROUTINE PROVAR(*)
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
      COMMON /FL/ VRB,VSUB,LC,LCP
      DATA LP/'('/,RP/')'/,COM/','/
C
      CALL NTBLNK
      CALL BLANK
      IF (KOL.EQ.80) RETURN 1
```

```
      DO 3260 I=1,3
      IF (STR(KOL).EQ.LP.OR.STR(KOL).EQ.RP.OR.STR(KOL).EQ.
     1 COM) KOL=KOL+1
      CALL BLANK
      CALL GTOPND
      VAR(I)=VRB
      VS(I)=VSUB
      CALL BLANK
 3260 CONTINUE
C
      RETURN
      END

      SUBROUTINE LETT
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
      COMMON /FL/ VRB,VSUB,LC,LCP
      DATA LP/'('/,RP/')'/,COM/','/
C
      DO 3370 I=1,3
      CALL GTOPND
      VAR(I)=VRB
      VS(I)=VSUB
      CALL BLANK
      IF (STR(KOL).EQ.LP.OR.STR(KOL).EQ.RP.OR.STR(KOL).EQ.
     1 COM) KOL=KOL+1
      CALL BLANK
      IF (KOL.EQ.80) RETURN
      CALL GETOTR
 3370 CONTINUE
C
      RETURN
      END

      SUBROUTINE GTOPND
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      COMMON /FL/ VRB,VSUB,LC,LCP
      DATA LL/'L'/,RR/'R'/,MM/'M'/
      DATA LP/'('/,RP/')'/,COM/','/
C
      IF (STR(KOL).EQ.LP.OR.STR(KOL).EQ.RP.OR.STR(KOL).EQ.
     1 COM) KOL=KOL+1
      CALL BLANK
      VARR=STR(KOL)
```

```
      IF (STR(KOL).EQ.LL) GO TO 3270
      IF (STR(KOL).EQ.RR) GO TO 3280
      IF (STR(KOL).EQ.MM) GO TO 3290
      VRB=4
      GO TO 3310
C
 3270 VRB=1
      GO TO 3300
 3280 VRB=2
      GO TO 3300
C
 3290 VRB=3
 3300 KOL=KOL+1
      CALL BLANK
      IF (STR(KOL).EQ.LP) GO TO 3320
      IF (VARR.EQ.LL) GO TO 3315
 3310 CALL GETNUM
      CALL GETCON
      VSUB=LCP
      GO TO 3340
 3315 VRB=4
      GO TO 3330
C
 3320 KOL=KOL+1
      CALL BLANK
      IF (VARR.EQ.LL.AND.STR(KOL).NE.LL) GO TO 3315
      IF (STR(KOL).NE.LL) GO TO 3310
 3330 CALL GETNUM
      VSUB=NBR
C
 3340 RETURN
      END


      SUBROUTINE MKLIN(KY,A,J,ORD)
      IMPLICIT INTEGER (A-Z)
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
C
      B=A+1
      DO 3350 I=1,4
      IF (VAR(J).EQ.I) GO TO (651,652,653,654),I
 3350 CONTINUE
C
  651 LIN(A)=VS(J)*10000+33
      GO TO (611,612),KY
  611 LIN(B)=320000+ORD
      GO TO 3360
  612 LIN(B)=ORD*10000+32
```

```
      GO TO 3360
C
  652 LIN(A)=VS(J)*10000+17
      GO TO (621,622),KY
  621 LIN(B)=160000+ORD
      GO TO 3360
  622 LIN(B)=ORD*10000+16
      GO TO 3360
C
  653 LIN(A)=VS(J)*10000+18
      GO TO (631,632),KY
  631 LIN(B)=180000+ORD
      GO TO 3360
  632 LIN(B)=ORD*10000+18
      GO TO 3360
C
  654 LIN(A)=C
      GO TO (641,642),KY
  641 LIN(B)=VS(J)*10000+ORD
      GO TO 3360
  642 LIN(B)=ORD*10000+VS(J)
C
 3360 RETURN
      END


      SUBROUTINE MKORD(LN)
      IMPLICIT INTEGER (A-Z)
      COMMON/FE/ VAR(3),VS(3),LIN(3),L(999),LA,OSW
C
      DO 3380 I=1,LN
      IF (LIN(I).EQ.0) GO TO 3380
      LA=LA+1
      L(LA)=LIN(I)
      CALL UPDATE(1,LA)
      IF(OSW.EQ.1) WRITE(6,3379) LA,L(LA)
 3379 FORMAT(50X,I3,2X,I8)
 3380 CONTINUE
C
      RETURN
      END


      SUBROUTINE GETCTR
      IMPLICIT INTEGER (A-Z)
      DIMENSION OPER(5)
      COMMON /FR/ STR(80),KOL,NBR,OPR
      DATA OPER/'+','-','*','/','='/
```

```
C
      DO 3470 I=1,5
      IF (STR(KCL).EQ.OPER(I)) GO TO 3480
 3470 CONTINUE
      RETURN
C
 3480 KOL=KCL+1
      OPR=I
      RETURN
      END

      SUBROUTINE GETCON
      IMPLICIT INTEGER (A-Z)
      COMMON /FR/ STR(80),KCL,NBR,CPR
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
      COMMON /FL/ VRB,VSUB,LC,LCP
      COMMON /FH/ DG
C
      DO 3390 I=LC,899
      IF (L(I).EQ.NBR) GO TO 3400
 3390 CONTINUE
C
      LC=LC-1
      IF (LC.EQ.LA) DG=1
      L(LC)=NBR
      CALL UPDATE(1,LC)
      LCP=LC
      RETURN
C
 3400 LCP=I
      RETURN
      END

      SUBROUTINE GETORD(LORD,LCPA,LCPB)
      COMMON/FE/ VAR(3),VS(3),LIN(8),L(999),LA,OSW
C
      LOPA=LCRD/10000
      LOPB=LCRD-(LCPA*10000)
      RETURN
      END

      SUBROUTINE GETNLU(N)
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 DSPLY
      COMMON/FE/ DSPLY(72,28),LOW,UP
```

```
      COMMON/FR/ STR(80),KOL,NBR,CPR
      DATA LL/'L'/,RR/'R'/,MM/'M'/,MUS/'-'/
C
      CALL NTBLNK
      CALL BLANK
      IF (STR(KCL).EQ.LL) N=1
      IF (STR(KCL).EQ.RR) N=2
      IF (STR(KCL).EQ.MM) N=3
      CALL GETNUM
      LOW=IABS(NBR)
      IF (STR(KCL).EQ.MUS) CALL GETNUM
      UP=IABS(NBR)
C
      RETURN
      END


      SUBROUTINE PRDIAG(D,*)
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 DSPLY,D1(24),D2(24),D3(24),D4(24),BLANK1
      LOGICAL*1 D5(24), D6(24)
      COMMON/FF/ DSPLY(72,28), LOW, UP
      COMMON/JP/ NULL,IGD3
C
      DATA BLANK1/' '/
      DATA D1/'E','N','D',' ','O','F',' ','D','A','T','A',
     2 ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '/
      DATA D2/'I','N','C','C','R','R','E','C','T',' ','O',
     2 'P','E','R','A','T','I','C','N',' ','C','O','D','E'/
      DATA D3/'M','I','C','R','O','-','M','E','M','O','S',
     2 'P','A','C','E',' ','E','X','C','E','E','D','E','D'/
      DATA D4/'E','N','D',' ','O','F',' ','F','I','L','E',
     2 ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '/
      DATA D5/'N','O','T',' ','A',' ','C','O','N','T','R',
     2 'O','L',' ','C','A','R','D',' ',' ',' ',' ',' ',' '/
      DATA D6/'U','N','R','E','C','O','G','N','I','Z','A','B',
     2 'L','E',' ','C','N','T','L',' ','C','A','R','D'/
C
      DO 8105 I=25,62
      DSPLY(I+10,27)=BLANK1
 8105 CONTINUE
C
      GOTO (8110,8120,8130,8140,8126,8128), D
C
 8110 DO 8115 I=1,24
 8115 DSPLY(I+10,27)=D1(I)
      GO TO 8150
C
```

```
 8120 DO 8125 I=1,24
 8125 DSPLY(I+10,27)=D2(I)
      GO TO 8150
C
 8130 DO 8135 I=1,24
 8135 DSPLY(I+10,27)=D3(I)
      GO TO 8150
C
 8140 DO 8145 I=1,24
 8145 DSPLY(I+10,27)=D4(I)
      GOTO 8150
C
 8126 DO 8127 I = 1,24
 8127 DSPLY(I+10,27) = D5(I)
      GOTO 8150
C
 8128 DO 8129 I = 1,24
 8129 DSPLY(I+10,27) = D6(I)
C
 8150 WRITE(6,8151) (DSPLY(I,27),I=11,72)
 8151 FORMAT(//1CX,80A1)
      CALL PTEXT(IGD3,DSPLY(1,27),72,27,NULL,3,40,230)
C
      RETURN 1
      END
//LINK.SYSIN DD *
       INCLUDE SYSLIB(IHCGSP03)
//GO.FT40F001 DD UNIT=(2250-1)
//GO.SYSIN DD *
```

LIST OF REFERENCES

1. International Business Machines Corporation Form C27-6932-2, IBM System/360 Operating System, Graphic Programming Services for FORTRAN IV, 1968.

2. International Business Machines Corporation Form C28-6515-5, IBM System/360 FORTRAN IV Language, 1966.

3. International Business Machines Corporation Form A27-2701-2 IBM System/360 Component Description IBM 2250 Display Unit Model 1, 1969.

4. Wilkes, M. V., "The Growth of Interest in Microprogramming: A Literature Survey" Computing Surveys 1, No. 3, September 1969.

5. Beck, L. and Keller, F., "The C-8401 Data Processor" Datamation 5, No. 2, p. 33-35, February 1964.

6. Boutwell, C. O., Jr., "The PB440" Datamation 5, No. 2, p. 30-32, February 1964.

7. Glantz, W. T., "A Note on Microprogramming", Journal of the Association for Computing Machinery 3, No. 2, p. 77-84, April 1956.

8. McGee, W. C., "The TRW-133 Computer" Datamation 5, No. 2, p. 27-29, February 1964.

9. Semarne, W. M. and Porter, R. E., "A Stored Logic Computer" Datamation 2, No. 5, p. 33-36, May 1961.

10. Tucker, S. G., "Microprogram Control for System/360" IBM Systems Journal 6, No. 4, p. 222-241, 1967.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chief of Naval Operations (OP-91) Department of the Navy Washington, D. C. 20350	1
4.	Asst. Professor G. Heidorn Code 55 Hd (thesis advisor) Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	17
5.	Ens. R. C. Bolles, Code 53 Bq Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
6.	Professor D. G. Williams, Code 0211 Computing Center Naval Postgraduate School Monterey, California 93940	1
7.	LT Alan W. Frelich, USN COMPHIBRON TWELVE FPO New York, New York 09501	1
8.	LT Michael C. Roth, USN Air Department U.S.S. JOHN F. KENNEDY (CVA-67) FPO New York, New York 09501	1

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
A Simulated Micro-Programmed Computer Utilizing the Graphic Display of an IBM 360

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*
Master's Thesis, December 1969

5. AUTHOR(S) *(First name, middle initial, last name)*
Alan W. Frelich
Michael C. Roth

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1969	117	10

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A small hypothetical computer was designed and simulated using a 2250 Display Unit operating on an IBM 360/67 computer. The hypothetical computer features a micro-programming capability which allows the user to design his own machine language structure for any particular application. The 2250 console is set up to simulate the operator's console of the hypothetical machine, with the CRT being used to display selected portions of memory and registers. For ease of use, a compiler is included in the system to allow the user to write his microprogram in a higher level language.

Discussions of the compiler, the interpreter, and the programming and creation of the proper type of graphic display are included. A set of operating instructions for the hypothetical computer enables the user to program and operate the machine without the need to be familiar with the 2250 display unit beforehand.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Microprogramming						
Compiler						
Interpreter						
Graphic Display						